AD

TECHNICAL REPORT ARCCB-TR-96010

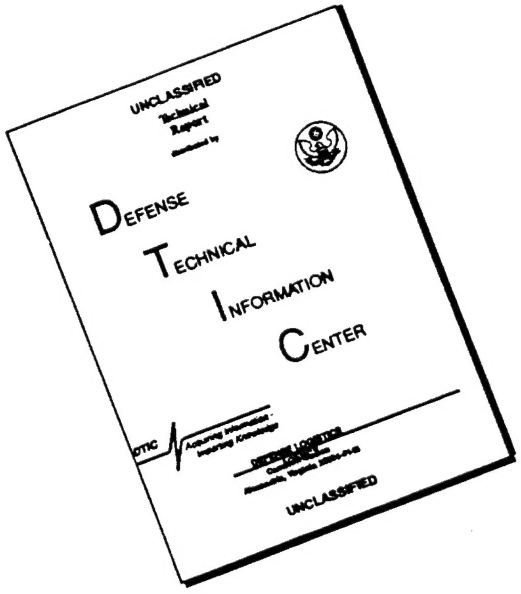
MATLAB® MODELING OF NON-UNIFORM BEAMS USING THE FINITE ELEMENT METHOD FOR DYNAMIC DESIGN AND ANALYSIS

ERIC L. KATHE

APRIL 1996

US ARMY ARMAMENT RESEARCH,
DEVELOPMENT AND ENGINEERING CENTER
CLOSE COMBAT ARMAMENTS CENTER
BENÉT LABORATORIES
WATERVLIET, N.Y. 12189-4050

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

DISCLAIMER

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

The use of trade name(s) and/or manufacturer(s) does not constitute an official indorsement or approval.

DESTRUCTION NOTICE

For classified documents, follow the procedures in DoD 5200.22-M, Industrial Security Manual, Section II-19 or DoD 5200.1-R, Information Security Program Regulation, Chapter IX.

For unclassified, limited documents, destroy by any method that will prevent disclosure of contents or reconstruction of the document.

For unclassified, unlimited documents, destroy when the report is no longer needed. Do not return it to the originator.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	April 1996	Final

4. TITLE AND SUBTITLE
MATLAB® MODELING OF NON-UNIFORM BEAMS USING THE FINITE ELEMENT METHOD FOR DYNAMIC DESIGN AND ANALYSIS

5. FUNDING NUMBERS
AMCMS No. 6226.24.H191.1

6. AUTHOR(S)
Eric L. Kathe

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army ARDEC
Benet Laboratories, AMSTA-AR-CCB-O
Watervliet, NY 12189-4050

8. PERFORMING ORGANIZATION REPORT NUMBER
ARCCB-TR-96010

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army ARDEC
Close Combat Armaments Center
Picatinny Arsenal, NJ 07806-5000

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The purpose of this report is to develop a model of non-uniform beams within a software environment that lends itself to design, simulation, and analysis of dynamic systems. The beam is modeled by the Euler-Bernoulli approximation using the finite element method. Reformulation of the second-order symmetric beam equations into the first-order state-space domain enables dynamic analysis and design from within the modern control paradigm. The MATLAB® software package is chosen as it is the current state-of-the-art modeling environment for control engineering. This report will present the development of the modeling software, develop some of the underlying theory and limitations of the analysis, demonstrate its use in the dynamic modeling and analysis of an XM291 gun system, and validate its results against known analytic results.

14. SUBJECT TERMS
MATLAB®, Non-Uniform Beam Dynamics, Frequency Response, Eigen Analysis, State-Space

15. NUMBER OF PAGES
91

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

MATLAB® Modeling of Non-Uniform Beams Using the Finite Element Method for Dynamic Design and Analysis

ABSTRACT

The purpose of this report is to develop a model of non-uniform beams within a software environment that lends itself to design, simulation, and analysis of dynamic systems. The beam is modeled by the Euler-Bernoulli approximation using the finite element method. Reformulation of the second-order symmetric beam equations into the first-order state-space domain enables dynamic analysis and design from within the modern control paradigm. The MATLAB® software package is chosen as it is the current state-of-the art modeling environment for control engineering. This report will present the development of the modeling software, develop some of the underlying theory and limitations of the analysis, demonstrate its use in the dynamic modeling and analysis of an XM291 gun system, and validate its results against known analytic results.

by

Eric L. Kathe
US Army Tank and Automotive Command
Benét Laboratories, AMSTA-AR-CCB-TC
Watervliet, NY 12189

LIST OF FIGURES

ACKNOWLEDGEMENTS

The author would like to acknowledge Dr. Andrew Lemnios —Rensselaer Polytechnic Institute, Troy, NY— Dr. Ronald Gast, Michael Gully, John Higgins Jr., Martin Leach, G. Peter O'Hara, Lawrence Rusch, Michael Soja, and Dr. Patrick Vottis —U. S. Army, Benét Laboratories, Watervliet Arsenal, NY— and Michael Mattice —U. S. Army, Fire Support Armaments Center, Picatinny Arsenal, NJ— for their assistance and insightful suggestions throughout this effort.

1 INTRODUCTION

The purpose of this report is to develop a dynamic model of non-uniform beams within the MATLAB® software environment (The MathWorks, Inc. / 24 Prime Park Way / Natick, MA 01760-1500). This will leverage MATLAB®'s capability in dynamic system design, analysis, simulation, and control formulation. The current application for this modeling is the dynamic analysis of the XM291 gun system, exclusive of the firing event. However, the formulation of the modeling was executed in a generic form to facilitate broad applicability to non-uniform beam dynamics.

Modeling of gun systems using the Euler-Bernoulli finite element technique to generate the second-order symmetric equations of motion with subsequent conversion to the first-order state-space domain has been accomplished by at least two previous authors. [1, 2] This approach does not lend itself to the analysis of the firing event due to the dependence of the ballistic loading on spatial derivatives of the mode-shape and continuity of load application. For this reason custom modeling techniques have been developed to address these issues. [3, 4] Lagrangian formulation of the second-order equations of motion with subsequent conversion to state-space has been accomplished for beams of uniform cross-section to promote analytic investigation of dynamic control issues without the burden of non-uniform beam approximation. [5]

This report will document the detailed development of the dynamic modeling approach and conduct several case studies including two of the XM291 gun system as an elastic beam. The computer files described in this report are listed in their entirety in the appendix.

2 THE DYNAMIC MODEL

2.1 THE FINITE ELEMENT BEAM MODELING METHOD

The means chosen to dynamically model non-uniform beams is the finite element method. This route has been chosen to provide a well known formulation that affords the opportunity to increase or decrease the complexity and subsequent accuracy of the discrete model approximation in a robust and predictable manner. This allows the analyst to conduct preliminary studies using low-order models to gain a perspective of gross dynamic properties that efficiently provides a focus to later conduct high accuracy analysis using larger models.

The finite element method employed in this report utilizes the Euler-Bernoulli beam approximation and the Hermite-cubic interpolation functions to form the inertial and stiffness matrices of the undamped second-order symmetric equations of motion. [6, 7] This is achieved by approximating the continuous non-uniform beam as an assemblage of a finite number of discrete elements. Within each discrete element, the interpolation functions are used to approximate the interior deformation. At the boundary between two adjacent elements, called a node, continuity of lateral displacement and slope are imposed. When assembled, the resulting finite element model dynamics, governed solely by the node states, closely approximates the dynamics of the non-uniform beam. In the limit, Simpson's hypothesis states that if a sufficient number of permissible finite elements are employed, the finite element modeling and the continuum modeling become equivalent. [8]

2.1.1 LIMITATIONS OF THE FINITE ELEMENT MODELING METHOD

The Euler-Bernoulli beam element used does not model shear effects, nor the cross-sectional rotational inertia of individual elements that are included in the more advanced —Timoshenko— beam equation. [6] These un-modeled dynamics play an increasing role as the beam's being modeled become less slender, and as the frequency response of interest increases. For gun geometries, it has been shown that these effects are relatively small in the dynamic region of interest, and may be neglected for most analysis. [9] This method also neglects to model axial mode dynamics such as would be experienced by the barrel, as a rod, in axial tension and compression. This effect could readily be added, and is outlined in the references given.

Finally, the Euler-Bernoulli formulation only models transverse vibrations in one plane, such as the vertical or horizontal planes of motion. Implicit in this assumption, is that the transverse motion of the beam in its two orthogonal planes will be decoupled, allowing the analyst to consider the dynamics in the two separate planes independently.

2.1.2 EQUATION OF MOTION FOR A SINGLE FINITE ELEMENT

The equation of motion for a single finite element will require the formation of two matrices to represent the inertial and stiffness parameters in a form that is a function of the elements nodal lateral displacement and slope and their second temporal derivatives. Since a single Euler-Bernoulli beam element has two nodes, one at each end, the inertial and stiffness matrices will be four-by-four in size. In order to facilitate the modeling of non-uniform beams, the formation of these matrices will explicitly use the Hermite-cubic interpolation functions as listed below where x is the linear position along the element and h is its total length: [6, 7]

$$\phi_1(x) = 1 - 3\left(\frac{x}{h}\right)^2 + 2\left(\frac{x}{h}\right)^3 \quad (a) \qquad \phi_3(x) = 3\left(\frac{x}{h}\right)^2 - 2\left(\frac{x}{h}\right)^3 \quad (c)$$

$$\phi_2(x) = x - 2h\left(\frac{x}{h}\right)^2 + h\left(\frac{x}{h}\right)^3 \quad (b) \qquad \phi_4(x) = -h\left(\frac{x}{h}\right)^2 + h\left(\frac{x}{h}\right)^3 \quad (d) \qquad \textbf{(1)}$$

The mass matrix can then be evaluated by integrating the linear density, $\rho(x)$, with the interpolation functions as follows: [6, 7]

$$\boldsymbol{M} = \begin{bmatrix} \int_0^h \rho(x)\phi_1(x)\phi_1(x)\,dx & \int_0^h \rho(x)\phi_1(x)\phi_2(x)\,dx & \int_0^h \rho(x)\phi_1(x)\phi_3(x)\,dx & \int_0^h \rho(x)\phi_1(x)\phi_4(x)\,dx \\ \int_0^h \rho(x)\phi_2(x)\phi_1(x)\,dx & \int_0^h \rho(x)\phi_2(x)\phi_2(x)\,dx & \int_0^h \rho(x)\phi_2(x)\phi_3(x)\,dx & \int_0^h \rho(x)\phi_2(x)\phi_4(x)\,dx \\ \int_0^h \rho(x)\phi_3(x)\phi_1(x)\,dx & \int_0^h \rho(x)\phi_3(x)\phi_2(x)\,dx & \int_0^h \rho(x)\phi_3(x)\phi_3(x)\,dx & \int_0^h \rho(x)\phi_3(x)\phi_4(x)\,dx \\ \int_0^h \rho(x)\phi_4(x)\phi_1(x)\,dx & \int_0^h \rho(x)\phi_4(x)\phi_2(x)\,dx & \int_0^h \rho(x)\phi_4(x)\phi_3(x)\,dx & \int_0^h \rho(x)\phi_4(x)\phi_4(x)\,dx \end{bmatrix} \tag{2}$$

Similarly the stiffness matrix may be formed from the material's elasticity, E(x), its second areal moment, I(x), and the second spatial derivative of the interpolation functions: [6, 7]

$$\boldsymbol{K} = \begin{bmatrix} \int_0^h E(x)I(x)\phi_1''(x)\phi_1''(x)\,dx & \int_0^h E(x)I(x)\phi_1''(x)\phi_2''(x)\,dx & \int_0^h E(x)I(x)\phi_1''(x)\phi_3''(x)\,dx & \int_0^h E(x)I(x)\phi_1''(x)\phi_4''(x)\,dx \\ \int_0^h E(x)I(x)\phi_2''(x)\phi_1''(x)\,dx & \int_0^h E(x)I(x)\phi_2''(x)\phi_2''(x)\,dx & \int_0^h E(x)I(x)\phi_2''(x)\phi_3''(x)\,dx & \int_0^h E(x)I(x)\phi_2''(x)\phi_4''(x)\,dx \\ \int_0^h E(x)I(x)\phi_3''(x)\phi_1''(x)\,dx & \int_0^h E(x)I(x)\phi_3''(x)\phi_2''(x)\,dx & \int_0^h E(x)I(x)\phi_3''(x)\phi_3''(x)\,dx & \int_0^h E(x)I(x)\phi_3''(x)\phi_4''(x)\,dx \\ \int_0^h E(x)I(x)\phi_4''(x)\phi_1''(x)\,dx & \int_0^h E(x)I(x)\phi_4''(x)\phi_2''(x)\,dx & \int_0^h E(x)I(x)\phi_4''(x)\phi_3''(x)\,dx & \int_0^h E(x)I(x)\phi_4''(x)\phi_4''(x)\,dx \end{bmatrix} \tag{3}$$

The generalized coordinate vector and its second temporal derivative are: [6, 7]

$$\underline{q} = \begin{bmatrix} y|_{x=0} \\ \theta|_{x=0} \\ y|_{x=h} \\ \theta|_{x=h} \end{bmatrix}, \quad \ddot{\underline{q}} = \begin{bmatrix} \ddot{y}|_{x=0} \\ \ddot{\theta}|_{x=0} \\ \ddot{y}|_{x=h} \\ \ddot{\theta}|_{x=h} \end{bmatrix}, \quad \textit{where} \quad \begin{cases} y \textit{ Denotes Lateral Displacement.} \\ \theta \textit{ Denotes Slope, } \left(\frac{dy}{dx}\right). \\ 0 \textit{ Denotes Left Node.} \\ h \textit{ Denotes Right Node.} \\ \ddot{n} \textit{ Denotes } \frac{d^2n}{dt^2}. \end{cases} \tag{4}$$

The generalized forces may be found in one of two ways. First, the distributed force, p(x), can be explicitly integrated with the interpolation polynomials. [6, 7] Second, it can be inferred by reversing the reaction forces that would be experienced by a simply supported beam element subjected to the loading

along its span. [10, 11] we have chosen to implement the later approach in the m-file, <fem_force.m>, but we will display the former approach below:

$$f = \begin{bmatrix} \int_0^h p(x,t)\phi_1(x)\,dx \\ \int_0^h p(x,t)\phi_2(x)\,dx \\ \int_0^h p(x,t)\phi_3(x)\,dx \\ \int_0^h p(x,t)\phi_4(x)\,dx \end{bmatrix} \tag{5}$$

The equation of motion for the undamped, Euler-Bernoulli beam element can now be written by combining equations (2) - (5): [6, 7]

$$M\ddot{q} + Kq = f \tag{6}$$

The approximate interior deflections may be determined by evaluating the interpolation functions of (1), augmented by the generalized coordinate vector of (4) as follows: [11]

$$y(x) \approx \phi_1(x)\, y|_{x=0} + \phi_2(x)\, \theta|_{x=0} + \phi_3(x)\, y|_{x=h} + \phi_4(x)\, \theta|_{x=h} \tag{7}$$

2.1.3 COMBINED MOTION OF MULTIPLE FINITE ELEMENTS

The equations of motion of adjacent finite elements are coupled together by the imposed continuity of lateral displacement and slope at the intersecting nodes. Consider a simple two element system composed of element one of length h_1 and element two of length h_2 joined in the middle. Inspection of the generalized coordinate vectors reveals that the later two generalized coordinates of element one, and the first two generalized coordinates of element two, represent the same boundary condition. Thus the number of generalized coordinates required to represent the two element structure has been reduced from eight to six, with a total of three nodes. Numbering the left-free node, one, the middle node, two, and the right free node, three, and superscripting the elemental matrices with the element number, we can combine the mass and stiffness matrices and the force and generalized coordinate vectors as follows: [6, 7]

$$M = \begin{bmatrix} M^1{}_{1,1} & M^1{}_{1,2} & M^1{}_{1,3} & M^1{}_{1,4} & 0 & 0 \\ M^1{}_{2,1} & M^1{}_{2,2} & M^1{}_{2,3} & M^1{}_{2,4} & 0 & 0 \\ M^1{}_{3,1} & M^1{}_{3,2} & \left(M^1{}_{3,3}+M^2{}_{1,1}\right) & \left(M^1{}_{3,4}+M^2{}_{1,1}\right) & M^2{}_{1,3} & M^2{}_{1,4} \\ M^1{}_{4,1} & M^1{}_{4,2} & \left(M^1{}_{4,3}+M^2{}_{2,1}\right) & \left(M^1{}_{4,4}+M^2{}_{2,1}\right) & M^2{}_{2,3} & M^2{}_{2,4} \\ 0 & 0 & M^2{}_{3,1} & M^2{}_{3,2} & M^2{}_{3,3} & M^2{}_{3,4} \\ 0 & 0 & M^2{}_{4,1} & M^2{}_{4,2} & M^2{}_{4,3} & M^2{}_{4,4} \end{bmatrix} \tag{8}$$

$$K = \begin{bmatrix} K^1{}_{1,1} & K^1{}_{1,2} & K^1{}_{1,3} & K^1{}_{1,4} & 0 & 0 \\ K^1{}_{2,1} & K^1{}_{2,2} & K^1{}_{2,3} & K^1{}_{2,4} & 0 & 0 \\ K^1{}_{3,1} & K^1{}_{3,2} & \left(K^1{}_{3,3}+K^2{}_{1,1}\right) & \left(K^1{}_{3,4}+K^2{}_{1,1}\right) & K^2{}_{1,3} & K^2{}_{1,4} \\ K^1{}_{4,1} & K^1{}_{4,2} & \left(K^1{}_{4,3}+K^2{}_{2,1}\right) & \left(K^1{}_{4,4}+K^2{}_{2,1}\right) & K^2{}_{2,3} & K^2{}_{2,4} \\ 0 & 0 & K^2{}_{3,1} & K^2{}_{3,2} & K^2{}_{3,3} & K^2{}_{3,4} \\ 0 & 0 & K^2{}_{4,1} & K^2{}_{4,2} & K^2{}_{4,3} & K^2{}_{4,4} \end{bmatrix} \tag{9}$$

$$q = \begin{bmatrix} y|_{x=0} \\ \theta|_{x=0} \\ y|_{x=h_1} \\ \theta|_{x=h_1} \\ y|_{x=(h_1+h_2)} \\ \theta|_{x=(h_1+h_2)} \end{bmatrix} = \begin{bmatrix} y_1 \\ \theta_1 \\ y_2 \\ \theta_2 \\ y_3 \\ \theta_3 \end{bmatrix} \quad (a) \,, \qquad f = \begin{bmatrix} f^1{}_1 \\ f^1{}_2 \\ \left(f^1{}_3+f^2{}_1\right) \\ \left(f^1{}_4+f^2{}_2\right) \\ f^2{}_3 \\ f^2{}_4 \end{bmatrix} = \begin{bmatrix} F_1 \\ M_1 \\ F_2 \\ M_2 \\ F_3 \\ M_3 \end{bmatrix} \quad (b) \tag{10}$$

Note that we have adopted a more convenient subscripted naming convention for the generalized coordinates and generalized force vector from the nodal reference. The equation of motion (6) is still valid for this larger, six-by-six, system of linear equations.

An indefinite number of additional elements can be combined in the same manner. The number of generalized coordinates will be twice the number of nodes, which equals the total number of elements plus one for a free-free beam model.

It is important to note that the inertial and stiffness matrices are symmetric. This has important implications, on the complex form of the roots of the differential equations, that cancels any possible imaginary content in the generalized coordinates. Thus, the symmetry of these matrices prevents the modeling from becoming non-physically realizable. This will be elaborated on in section 3.5.2 in the context of section 3.2.

2.2 MATLAB® REALIZATION OF THE FINITE ELEMENT BEAM MODEL

MATLAB® m-files were written to implement the finite element beam model of the previous section. M-files are user written program code in the MATLAB® language that provide extensibility to MATLAB®. [12] All of the m-files written are function files. Function files provide for the passing of arguments from other work-spaces to the local function file work-space and back again. This prevents the over use of one work-space, thus reducing confusion of the available variables and the available variable names. Function files are also said to be compiled within MATLAB®, thus increasing efficiency. Finally, this format provides for the development of on-line documentation describing the function file and its operation. All explicit MATLAB® functions in addition to the m-files and variables are enclosed by angle brackets to distinguish them from regular text.

2.2.1 UNITS

Any consistent system of units may be used by this software. Automatic label generations assume that the linear frequencies computed are in units of Hertz. The meter-kilogram-second or international system of units (SI) is used in the input m-files. The poundal or slug consistent English systems, or the centimeter-gram-second (CGS) system could also be used.

2.2.2 INPUT BEAM GEOMETRY AND MATERIAL PROPERTIES

The first requirement to realizing the finite element model of a beam is to input the geometry and material properties. The required form of this data will consist of four vectors, of equal length, representing values of the beam associated with a fine-resolution and equidistant sampling of the axial position, and the respective linear density, stiffness, and non-beam linear density values. (The non-beam density values provide for the incorporation of extraneous masses whose elasticity is supposed to play a negligible role in the beam dynamics, but whose inertia can not be ignored.) For the case of the XM291 gun barrel, the geometric information (inner and outer radii) was available off of a finish-machine drawing in a format that lent itself to sampling every millimeter from an axial position of one millimeter to the length of the barrel. From the geometric information, the cross-sectional stiffness of the barrel at each axial position could readily be computed using MATLAB®'s array multiplication feature, and Young's modulus for the gun steel. The non-beam masses, such as the muzzle reference mount could then be included. An effort was made to distribute the non-beam masses over a significant axial length to prevent large fictitious spikes in the linear density values. In the absence of a non-beam linear density at a given axial location, this data vector contains zeros to maintain a one-to-one correspondence with the axial position vector.

The function file written to generate the four matrices for the XM291 gun system is named <geomf_XM291.m>. The names of the four data vectors are <spatial>, <lden>, <lEI>, and <lnbden> respectively. The length of the vectors is 6,750 which provides ample resolution of the geometry.

In addition to the four data vectors the file also produces a pair of two-by-two matrices formulated to allow for external rigid masses to be constrained to the free node's generalized coordinates at either end of the beam. This was required to allow for the incorporation of masses beyond the ends of the beam, such as the breech, which will be discussed in section 2.3.2.

An additional output of <geomf_XM291.m> is a matrix, <gm>, whose two columns consist of the inner and outer radii respectively. This is useful for later plotting and animation.

A final output of <geomf_XM291.m> is an option for automatic plot generation, with an input title label, that depicts the geometry, non-beam mass location, total beam mass, total non-beam mass, and the linear density and stiffness as a function of the axial position vector. This output is particularly useful for input data validation. Its use is demonstrated for the XM291 in section 4.1.2 and for a sixty-millimeter hybrid gun in section 4.3.1.

In an effort to reduce input errors, and facilitate future input file generation, three support function files were written. The first, <geom_seg.m>, converts the measurements of radii at two axial locations into uniform tapers that correspond to the axial position vector. The second, <geom_nbseg.m>, distributes the non-beam mass over a specified portion of the axial position vector. The third, <geom_check.m>, verifies the one-to-one correspondence between the axial position vector and any of the three other data vectors. It also checks to be sure that the axial position vector is evenly spaced from the first increment to the final length. Finally, it checks to be sure all data values are positive, and it imposes column structure on the vectors. This third file is used throughout the modeling software to check the validity of input data.

2.2.3 GENERATION OF A FINITE ELEMENT MESH

Once the four data vectors are available for finite element formulation, they must be broken up into the desired number of elements. Further, it will become extremely important, as the modeling effort continues, to place nodes at specific locations where external forces may interact with the beam model. This is particularly true of the trunnion location, about which the gun barrel pivots as it is elevated. The function file <fem_mesh.m> was written to perform the function of automatic node location.

The <fem_mesh.m> file uses the four input data vectors, a vector of imposed node locations, and the desired number of finite elements to formulate a meshing vector. Like the function <geomf_XM291.m>, an optional argument automatically generates a plot of the final mesh, as a function of the axial position vector and the metric used to divide up the elements that is discussed below. The use of this option is demonstrated in section 4.1.3. It also utilizes the <geom_check.m> file to validate the input data.

In order to automate the process, a metric had to be identified that would provide a measure of desired uniformity between elements. In other words, a quantitative measure to be used to evenly space the node locations. Axial position was not chosen because it would not compensate for differences in stiffness and density between elements. The ratio of stiffness to linear density —along the data vectors— was chosen to increase the meshing density along portions of the beam where the stiffness was low, and the inertia was high, which would lead to increased dynamic activity in the lower modes of interest. Formulation of this metric was greatly facilitated by the built-in MATLAB® command, <cumsum>. [12]

Once a metric is identified, the meshing becomes a four stage process. First, the imposed node, locations must be determined. Second, the imposed nodes form an integer number of *super* elements between them. Thus, the number of finite elements that remain after the imposing the nodes must be determined. These are called *free* elements, because they are the elements that may be placed by the metric. Third, the ideal integer number of *free* elements allotted to each *super* segment must be determined from the metric. Finally, each super element must be broken up by the metric to include its allotted number of *free* elements.

In addition to the above routine, three checks are executed to ensure that the number of nodes assigned to any span of the beam does not exceed the spatial resolution of the axial position vector. (To do so would result in collocated nodes.) If this situation is detected, <while> loops [12] are employed to identify spans of the beam that could support additional nodes, and redistribute the collocated nodes as necessary. Since these loops are dangerous in the sense that a small programming error could result in an infinite loop, a warning is written to the screen that the loop has been entered. In general, this situation is only encountered when a non-uniform beam tapers down to a point, which often leads to the stiffness going to zero faster than the density as in the case presented in section 4.4.1.

2.2.4 ELEMENTAL MASS AND STIFFNESS MATRIX FORMULATION

Realization of the elemental mass and stiffness matrices requires the computation of equations (1), (2), and (3) for a given element. This was achieved using the two files, <fem_interp.m> and <fem_beamel.m>, to numerically compute the interpolation functions and execute the matrix element integrations respectively. Both files use the <geom_check.m> file, discussed in section 2.2.2, to validate input data format which consists of three data vectors: axial position, combined beam and non-beam linear density, and cross-sectional stiffness. Like the axial position vector of the entire beam, the elemental position vector begins at the first increment of measure (*near* zero), and continues to the total length of the element, h.

The interpolation function of equation (1) and their second spatial derivatives as required by equation (3) were explicitly generated using the Symbolic Math TOOLBOX [13] although they can readily be evaluated by hand. These symbolic equations are then numerically evaluated at each axial position along the element using MATLAB® 's array operators and the <eval> command. [12] The output of <fem_interp.m> consists of two matrices consisting of four columns, one for each interpolation function, with the row elements forming a one-to-one correspondence with the axial position vector. The first matrix includes the interpolation functions, while the second matrix contains their second spatial derivatives.

The file, <fem_beamel.m>, numerically evaluates the integrals of equations (2) and (3) as a simple matrix/inner product scaled by the sampling resolution. (This is equivalent to a Riemann sum of the multiplied sampled data vectors.) This works well as long as the length of the interpolation vectors are long enough for the integration errors to become small. This accuracy issue is enforced by expanding the input vectors, using a zero-order-hold approach while increasing the spatial resolution, if the length of interpolation functions would otherwise fall below a set value. We have chosen to set the value at fifty. For pragmatic reasons, if the data expansion is required, the new data vector length is the minimum integer multiple of the former length that will meet or exceed the set minimum length. (Note that this condition may be an indication that a finer resolution of the input data is required. This is not addressed by the zero-order-hold data expansion. This is especially true if it is occurring with a relatively small number of finite elements.)

The complete output of these two files are numerical approximations of the four-by-four inertial and stiffness matrices of equations (2) and (3).

2.2.5 SYSTEM MASS AND STIFFNESS MATRIX FORMULATION

Once the elemental mass and stiffness matrices have been formed, they must be combined as in equations (8) and (9) to form the integrated system model. This is achieved by the <fem_form.m> file. The inputs to this file consist of the four raw data vectors of axial position and corresponding linear density, cross-sectional stiffness, and non-beam density. Further, this file requires the node location vector (mesh) generated by the file <fem_mesh.m>. Finally, a provision is made for the automatic generation of images of the matrices for qualitative model validation if an optional argument is passed. The use of this feature is demonstrated in section 4.1.4.

In addition to the use of the m-file <geom_check.m>, discussed in section 2.2.2, to validate the four data vectors, a separate file, <fem_node_check.m>, was written to validate the node vector with respect to the axial position vector prior to finite element formulation. This file insures that the node locations fall within the beam geometry, and that no node locations are repeated. Finally, it imposes a sorted column structure on the vector. This file also uses the <geom_check.m> file to insure that the axial position vector is valid.

The m-file <fem_form.m> first adds the beam, and non-beam mass vectors into a single combined linear density vector. It then sequentially breaks up the beam into the finite elements specified by the node coordinate vector. The elemental mass and stiffness matrices are formulated using the <fem_beamel.m> and <fem_interp.m> files described in section 2.2.4.

For efficiency, the system mass and stiffness matrices are initialized as square matrices, filled with zeros, of their final n-by-n size where n is the number of generalized coordinates. (The number of generalize coordinates is twice the number of nodes, including the free ends. Also note that the total number of nodes is equal to one plus the number of elements for the free-free beam model.) The system matrices are sequentially filled by the elemental matrices,. The intersections of the elemental matrices, as shown in the middle two-by-two sub-matrices of M and K of equations (8) and (9), are achieved by adding each new elemental matrix to the system matrix as it is being formed. This implementation cascades the addition of the elemental matrices down the main diagonal of the system matrices.

2.2.6 EQUIVALENT NODAL FORCE VECTOR FORMULATION

The system force vector of equation (10) is computed in a manner similar to the formulation of the system matrices from the elemental matrices. The file, <fem_force.m>, that realizes the system force vector requires four input vectors. The first vector consists of the axial position vector. The second two vectors consist of the point lateral force and point moment vectors that correspond one-to-one with the axial position vector in complete analogy with linear density vector. Distributed loads are represented by a series of equivalent point loads, down the force vector in a straight forward manner. The final required vector is the meshing vector of node index locations generated by <fem_mesh.m>. An optional input provides for explicit printing of the resulting system vector.

First the file computes the generalize forces and moments for the arbitrary loading of individual elements. As stated in section 2.1.2, the method used is to compute and sum the reverse reaction forces that would be experienced by a simply supported beam element subjected to the point loads applied by each input force value at its associated axial position.

The file then overlaps the generalized forces of adjacent elements at mutual node locations, in a manner similar to the cascading of the elemental matrices down the system matrices as described in section 2.2.5. This creates the generalize force vector as shown in equation (10).

This completes the numerical formulation of the finite element beam model governed by the dynamics of equation (6) with the full system inertial and stiffness matrices.

2.3 CONSTRAINT OF THE FINITE ELEMENT BEAM MODEL

The finite element model developed by equations (1) through (10) results in the free-free configuration. This means that the beam is not constrained to an inertial reference in any way. Most beams are constrained to an inertial reference in some manner. For the case of the XM291 gun barrel, one approximation would be to constrain its vertical beam vibrations at the gun trunnions (horizontally opposed pivot points), and at the elevation mechanism (the linear actuator that applies vertical forces to the breech end of the barrel, behind the trunnions). Unless otherwise stated, we will assume that the elevation mechanism is secured near the rear of the barrel by a stiff rod coupled to the inertial reference frame and the barrel by pin joints. (Inclusion of the elevation mechanism dynamics can later be included, but are not a part of this modeling effort. [1])

A related issue to the inclusion of constraints to the nodes of the finite element model is the inclusion of external inertias that are not adequately modeled by merely adding linear density to the beam model. This will be discussed later, in section 2.3.2.

Note that as stated in section 2.2.3, imposing node locations at the points of constraint is critical to the analysis of the system dynamics. If the external forces imposed on the beam were to occur between node locations, equations (5) and (10) would have to be employed to achieve the equivalent constraint. This extra step would complicate later dynamic analysis.

2.3.1 IMPOSED ESSENTIAL AND NATURAL BOUNDARY CONDITIONS

Two constraints may prevent the two rigid body modes associated with a beam element; translation and rotation. (This will be discussed in greater detail in section 3.4.1.) They may be imposed in one of three ways. The first two approaches assume that the points of constraint (the trunnion and elevation locations in the case of a gun barrel) are rigid, thus forming geometric constraints (essential boundary conditions). In the case of a gun system, lateral motion of the beam at the trunnion mechanism would be eliminated, while rotational motion (slope) at this point would be allowed.

The first approach is to algebraically eliminate the constrained generalized coordinate and its associated rows and columns from the mass and stiffness matrices. [2, 6] This reduces the size of the system equation (6) by one for each coordinate eliminated.

The second approach is to set the rows and columns of the mass and stiffness matrices associated with the constraint generalized coordinate to zero, with the exception of the diagonal term which is set to one. [11] In addition, the corresponding force vector element is set to zero. This method maintains a constancy of the generalized coordinate structure, (alternating y's and θ's) thus reducing confusion.

The third approach is to treat the locations as imposing constraint via elastic restoring forces (natural boundary conditions). This is the method adopted by Benét's uniform segment modeling (USM) code. [3, 9] The simplest restoring force is a constraint in the form of a linear elastic spring.

The third approach is the one adopted for the remainder of this modeling effort and is developed further in section 2.3.3.

2.3.2 COUPLED RIGID BODY MASS

Rigid body masses may be directly coupled to the generalized coordinates at node locations. This is particularly useful to incorporate point masses whose center of gravity does not lie within the geometry of the beam. It may also be used to incorporate non-beam elements with a significant rotary inertia that would be poorly approximated by a *thin* linear density.

The effect of an over-hanging mass is an increased system inertia, at the associated node. Due to the separation of the center of gravity of the rigid body mass from the node, the lateral deflection of the center of the rigid body mass will be a function of both the deflection and slope of the node. Rotational inertia will be formed by the offset, as could be predicted by the parallel axis theorem. Denoting the offset as r, which is positive when the rigid body of offset to the right of the associated node number, denoted by the subscript nn, and denoting the motion of the rigid mass about its center of gravity with the subscript, cg, we can write:

$$\begin{aligned} x_{cg} &= x_{nn} + r \\ y_{cg} &= y_{nn} + r\theta_{nn} \\ \theta_{cg} &= \theta_{nn} \end{aligned} \qquad \textit{where} \qquad \begin{cases} \dot{x}_{cg} = \dot{x}_{nn} = \dot{r} = 0 \{ & \textit{Small angle aproximation.} \\ \dot{y}_{cg} = \dot{y}_{nn} + r\dot{\theta}_{nn} & \succ \ \ddot{y}_{cg} = \ddot{y}_{nn} + r\ddot{\theta}_{nn} \\ \dot{\theta}_{cg} = \dot{\theta}_{nn} & \succ \ \ddot{\theta}_{cg} = \ddot{\theta}_{nn} \end{cases} \qquad \textbf{(11)}$$

The equation of motion for the rigid mass about its own center of gravity is:

$$\begin{aligned} F_{cg} &= Mass_{RB}\ddot{y}_{cg} \\ M_{cg} &= J_{RB}\ddot{\theta}_{cg} \end{aligned} \qquad \textbf{(12)}$$

Combining equations (11) and (12), the rigid body forces at the center of mass can be expressed in terms of the nodal generalized coordinates. In addition, the reaction forces (opposite to the direction of the rigid body mass acceleration) experienced by the node location may be expressed, noting that the offset creates an additional term for the nodal moment:

$$\begin{aligned} F_{cg} &= Mass_{RB}\left(\ddot{y}_{nn} + r\ddot{\theta}_{nn}\right) \quad (a) & \qquad F_{nn}^{React} &= -F_{cg} \quad (c) \\ M_{cg} &= J_{RB}\ddot{\theta}_{nn} \quad (b) & \qquad M_{nn}^{React} &= -M_{cg} - rF_{cg} \quad (d) \end{aligned} \qquad \textbf{(13)}$$

Combining terms, the reaction force that would emulate the rigid body inertia can be formulated as a two-by-two matrix (Note that the parallel axis theorem is incorporated into the lower right matrix element):

$$\begin{bmatrix} F_{nn}^{React} \\ M_{nn}^{React} \end{bmatrix} = - \begin{bmatrix} M_{RB} & rM_{RB} \\ rM_{RB} & \left(J_{RB} + r^2 M_{RB}\right) \end{bmatrix} \begin{bmatrix} \ddot{y}_{nn} \\ \ddot{\theta}_{nn} \end{bmatrix} \qquad \textbf{(14)}$$

These force terms can be directly combined with the force vector of equation (10b). Finally, by placing the additional force vector terms into the perspective of equation (6) it is clear that by moving the components of equation (14) to the left side of equation (6), the two-by-two matrix of (14) can be directly added to the corresponding two-by-two sub-matrix of the system mass matrix. (Note that the minus sign in front of the rigid body inertia matrix of (14) is negated by moving to the left side of equation (6).)

2.3.3 COUPLED EXTERNAL SPRINGS

External springs, that provide natural constraints upon the beam model, may be integrated into the finite element formulation in much the same way as the external rigid body mass. Springs, however, are easier to envision because they are collocated with the node. (The node location must be imposed in the problem formulation.)

The effect of a linear translational or rotational spring is that they opposes lateral or rotational deflections respectively, about the equilibrium location of the nodal coordinates. This opposition is in the form of a force or moment, opposite in direction, and proportional in magnitude to the deflection. Denoting lateral springs as K_y and rotational springs as K_θ where either K must be positive, and using the same convention of denoting the node number by the subscript nn, we can write the equivalent reaction forces as:

$$\begin{aligned} F_{nn}^{React} &= - K_y y_{nn} \qquad (a) \\ M_{nn}^{React} &= - K_\theta \theta_{nn} \qquad (b) \end{aligned} \qquad \textbf{(15)}$$

As before, either of these force terms can be directly combined with the force vector of equation (10). By placing the additional force vector terms into the perspective of equation (6) it is clear that by moving the reaction forces of (15) to the left side of equation (6), the positive scaler spring constants of (15) can be directly added to the corresponding diagonal element of the system stiffness matrix.

2.3.4 COUPLED EXTERNAL VIBRATION ABSORBERS

A very interesting dynamic effect can be achieved by coupling an external lumped mass-spring-dashpot system to dynamic systems. Often called a vibration absorber [14, 15] (or a mass-tuned-damper [16] in the civil engineering domain), the effect of the external system is to attenuate combined system vibration in a narrow band near the operating frequency of the absorber for sharp transients, and to dissipate steady-state energy across a wider band, if the damping coefficient is significant. They also present the possibility of additional, undesirable resonances. For the sake of simplicity, only the development of a lateral vibration absorber will be presented, although the rotational absorber is completely analogous.

As in the earlier coupling cases, it is critical for the point of coupling to coincide with a node location. This must be done during the formulation of the finite element model. As stated, the vibration absorber consists of a mass, a spring, and a dashpot. Since damping has not yet been introduced to the finite element beam model, the damping effect of the absorber will be ignored for now and set to zero. It will be incorporated later in section 2.4.2.

With the addition of the new mass, a new energy storing device has been added to the total system. This will require the inclusion of a new generalized coordinate to represent the deflection of this mass, and its time derivatives, from its equilibrium position. A convenient position to place the new coordinate, y_{VA}, is at the base of the generalized coordinate vector of (10a). The equation of motion for the absorber constrained to node, nn, is:

$$M_{VA}\ddot{y}_{VA} + K_{VA}\left(y_{VA} - y_{nn}\right) = 0 \qquad \textbf{(16)}$$

Clearly, the absorber will also cause a reactive force to be exerted on the associated node via the spring coupling. If placed on the right side of equation (6), in analogy with previous formulations, the lateral reaction force acting on node, nn, is:

$$F_{nn}^{React} = - K_{VA}\left(y_{nn} - y_{VA}\right) \qquad \textbf{(17)}$$

To incorporate equations (16) and (17) into the system matrices of (8) and (9) will require the expansion of each system matrix by a new row of zeros at the bottom and a new column of zeros on the right side to match the new element, y_{VA}, at the base of the generalized coordinate vector of (10a).

The inclusion of the inertia of the absorber can be completed by adding its mass, M_{VA}, to the zero in the lower right hand corner of the expanded system mass matrix.

The inclusion of the elasticity of the absorber will require four additions to the expanded system matrix. For clarity, the matrix row that corresponds to y_{nn} is denoted by $index_n$, and the new row that corresponds to y_{VA} is denoted by $index_{VA}$. (Since we are restricted to the lateral coordinate, one can reason that $index_n$ is equal to twice its node number minus one. Also, for one absorber, $index_{VA}$ is equal to twice the total number of nodes plus one.)

The bottom rows of the expanded system matrices can clearly be set using (16). The nodal row of the expanded stiffness matrix, corresponding to lateral deflection, can be modified using the right side of (17) by dropping the minus sign and adding in the new values to the existing values from the finite element formulation.

The single scaler inertial operation is represented below followed by all four scaler operations on the stiffness matrix that are condensed into a single two-by-two matrix representation. In both cases, the addition of the new scalars to the zeros of the new rows and columns is not explicitly shown. Only one case of an interior element modification occurs, and that is the interior element of the finite element

stiffness matrix, due to (17). For increased clarification, the previous value of this interior element is superscripted with the letters FEM to indicate that it is the value prior to the absorber modification:

$$M(index_{VA}, index_{VA}) = M_{VA} \tag{18}$$

$$\begin{bmatrix} K(index_n, index_n) & K(index_n, index_{VA}) \\ K(index_{VA}, index_n) & K(index_{VA}, index_{VA}) \end{bmatrix} = \begin{bmatrix} \left(K^{FEM}(index_n, index_n) + K_{VA}\right) & -K_{VA} \\ -K_{VA} & K_{VA} \end{bmatrix} \tag{19}$$

Once again, it is important to note that all of the above modifications maintain symmetry of the system matrices as was discussed in section 2.1.3.

2.4 SYSTEM DAMPING

All mechanical systems dissipate energy through motion. Inclusion of this effect within the matrix modeling paradigm is centered around formulating the damping in a matrix form compatible with (6) where the new damping matrix, C_D, is multiplied by the first time derivative of the generalized coordinates. [14, 17]

$$M\ddot{q} + C_D\dot{q} + Kq = f \tag{20}$$

The formulation of C_D is most commonly achieved via the Rayleigh damping approximation below. [14, 17]

$$C_D = \alpha M + \beta K \tag{21}$$

An important property of Rayleigh damping is that increases in α increase the damping due to inertia and thus effect greater damping in the lower frequencies. Conversely, increases in β effect greater damping in the higher frequencies. [15]

The coefficients, α and β, can be defined by choosing the critical damping ration, ζ, for two modes, j and k, as shown in (22). [1, 18] Complete control of the assignment of damping values for each vibratory mode can be achieve through modal transformation. [19]

$$\begin{bmatrix} \alpha \\ \beta \end{bmatrix} = 2\begin{bmatrix} 1 & \omega^2{}_j \\ 1 & \omega^2{}_k \end{bmatrix}^{-1}\begin{bmatrix} \omega_j\zeta_j \\ \omega_k\zeta_k \end{bmatrix} \tag{22}$$

Rayleigh damping, as shown in (21) will be used for the remainder of this modeling effort with the exception of external damping sources as outline below. When external damping sources are include, they will be added after the damping matrix formed by (21) is computed.

2.4.1 COUPLED EXTERNAL DASHPOTS

The effect of an external dashpot is completely analogous to the coupling of external springs. Linear translational or rotational dashpots oppose the relative lateral or rotational velocities respectively, between the inertial reference frame and the location of the nodal coordinates. This opposition is in the form of a force or moment, opposite in direction, and proportional in magnitude to the relative velocity. Denoting lateral dashpots as C_y and rotational dashpots as C_θ, where either C must be positive, and using the same convention of denoting the node number by the subscript nn, we can write the equivalent reaction forces as:

$$\begin{aligned} F_{nn}^{React} &= -C_y \dot{y}_{nn} \qquad (a) \\ M_{nn}^{React} &= -C_\theta \dot{\theta}_{nn} \qquad (b) \end{aligned} \tag{23}$$

As before, the positive scaler damping constant of (23) can be directly added to the corresponding diagonal element of the system damping matrix in (20).

2.4.2 VIBRATION ABSORBER DAMPING

Explicit inclusion of the damping of vibration absorber damping is shown below. It follows in complete analogy with the stiffness matrix modifications of (19) in light of the distinctions demonstrated in section 2.4.1.

$$\begin{bmatrix} C_D(index_n, index_n) & C_D(index_n, index_{VA}) \\ C_D(index_{VA}, index_n) & C_D(index_{VA}, index_{VA}) \end{bmatrix} = \begin{bmatrix} \left(C_D^{FEM}(index_n, index_n) + C_{VA}\right) & -C_{VA} \\ -C_{VA} & C_{VA} \end{bmatrix} \tag{24}$$

2.5 MATLAB® COUPLING OF EXTERNAL LUMPED PARAMETER ELEMENTS

The file <fem_lump.m> and an ancillary support file, <fem_lumpm_check.m>, were written to integrate external lumped elements and Rayleigh damping with the finite element model of the beam.

The files first add the two-by-two coupled rigid body terms to the finite element mass matrix. (No current provision exists for inclusion of rigid body elements at interior node locations, although this would be a simple programming exercise.) As stated in section 2.3.2, the main intention of this provision is to include the inertial effects of elements that are coupled to the beam, but whose centers of gravity lay beyond the axial beam geometry. This is required to model the inertia of the breech in the case of the XM291 gun system.

Rayleigh damping is then computed using the input parameters, α and β, as developed in (21) and the finite element matrices augmented by the rigid body inertia. This results in the three system matrices of equation (20).

External springs and dashpots are then integrated with the three system matrices as developed in (15) and (23). This is accomplished by passing a matrix of constraint parameters. Each row of the constraint parameter matrix contains the explicit generalized coordinate number, $index_n$, to which the constraints are coupled, the lumped spring constant, (force/displacement), and the damping constant, (force/velocity). (Note the generalized coordinate index is found using the convention of equation (10): $index_n = 2 \times nn - 1$ for translational motion and $index_n = 2 \times nn$ for rotational motion, where nn is the node number.) The number of rows indicates the number of external couplings. Note that these damping constants are conveniently computed using the same Rayleigh damping parameters as the matrices. It is implemented separately to provide the flexibility to incorporate non-Rayleigh damped external constraints such as hydraulic shock-absorbers. (Also note, MATLAB® matrices must be fully populated. Therefore, if a pure damping constraint is to be coupled, a spring constant of zero must be included in the row of the constraint parameter matrix.)

Finally, an optional input matrix will couple any lumped vibration absorbers with the three system matrices. This is the only operation that will change the size of the original finite element matrices. The absorber is accomplished by passing an absorber parameter matrix, that is identical to the external spring and dashpot matrix, with the addition of a fourth column that includes the inertia of the coupled devise (mass). This is done to effect the modifications of equations (18), (19), and (24). (Note: the terminology in the m-files refers to the vibration absorbers as mass tuned dampers, or MTD's, for reasons discussed in section 2.3.4.)

The final output of the files are the three system matrices that form equation (20).

2.6 STATE-SPACE FORMULATION FROM THE SECOND ORDER SYSTEM

Many of the powerful MATLAB® tools for dynamic analysis and design require the dynamic equations to be in linear, time-invariant, first-order state-space form. Equation (20) is in the second-order symmetric form. This representation of the system dynamics may be converted to the first-order state-space form by the following method: [2, 20]

First, define the state-vector, x, and its first time derivative as the combined generalized coordinates of (10a), possibly modified by the inclusion of vibration absorbers, and their first temporal derivatives:

$$x = \begin{bmatrix} q \\ \dot{q} \end{bmatrix} \rightarrow \dot{x} = \begin{bmatrix} \dot{q} \\ \ddot{q} \end{bmatrix} \tag{25}$$

Second, define the system dynamics of (20) in terms of the generalized coordinate vector's time derivatives:

$$\begin{aligned} \dot{q} &= I\,\dot{q} && (a) \\ \ddot{q} &= -M^{-1}Kq - M^{-1}C_D\dot{q} + M^{-1}f && (b) \end{aligned} \tag{26}$$

Note that the form of equation (26b) presumes that the mass matrix is invertible. This is always the case for beam finite element formulations. (A singular mass matrix would imply that at least one beam element had zero mass.) Using (25) and (26), the state-space representation with the generalized coordinates as the output is:

$$\begin{aligned} \dot{\underline{x}} &= \boldsymbol{A}\underline{x} + \boldsymbol{B}\underline{u} \\ \underline{y} &= \boldsymbol{C}\underline{x} + \boldsymbol{D}\underline{u} \end{aligned} \tag{27}$$

Where the state-space matrices are constructed in terms of the second order system matrices, and the zero and identity matrices of compatible size. (Note, the state-space matrices have twice the number of rows and columns of the second-order system matrices.):

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{I} \\ \left(\boldsymbol{M}^{-1}\boldsymbol{K}\right) & \left(\boldsymbol{M}^{-1}\boldsymbol{C}_D\right) \end{bmatrix} \quad (a) \qquad \boldsymbol{B} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{M}^{-1} \end{bmatrix} \quad (b)$$

$$\boldsymbol{C} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \end{bmatrix} \quad (c) \qquad \boldsymbol{D} = \begin{bmatrix} \boldsymbol{0} \end{bmatrix} \quad (d)$$

$$\underline{y} = \underline{q} \quad (e) \qquad \underline{u} = \underline{f} \quad (f) \tag{28}$$

2.7 MATLAB® REALIZATION OF THE STATE-SPACE FORMULATION

The m-file, <fem2ss.m>, computes the state-space matrices using the second-order matrices of (20) as shown in (28).

3 EIGEN ANALYSIS

3.1 UNDAMPED SECOND-ORDER EIGENVALUES AND EIGENVECTORS

A powerful perspective system dynamics can be attained by converting the dynamic representation of the vibrating system to the frequency or modal domain. It has been shown that the free vibrations of the homogeneous form of the differential equation of (6) (provided M and K are symmetric) can be solved using the separation of variables technique. [6] This method separates the spatial and temporal domains of the generalized coordinate vector.

$$\underline{q}(t) = \sum_{i=0}^{n} \underline{\Phi}_i \varphi_i(t) = \boldsymbol{\Phi}\underline{\varphi}(t) \tag{29}$$

This achieves a pairing of spatial shapes, the columns of Φ, with the dynamic amplitudes of the elements of $\underline{\varphi}(t)$. Equation (29) has decoupled the system into a series of single degree of freedom vibrating modes, where each associated mode shape represents the single degree of freedom. From this perspective, the dynamic amplitudes of $\underline{\varphi}(t)$ that form the solution of the differential equation can be formulated as:

$$\underline{\varphi}(t) = \begin{bmatrix} A_1\cos(\omega_1 t + \phi_1) \\ A_2\cos(\omega_2 t + \phi_2) \\ \cdot \\ \cdot \\ \cdot \\ A_n\cos(\omega_n t + \phi_n) \end{bmatrix} \succ \dot{\underline{\varphi}}(t) = \begin{bmatrix} -A_1\omega_1\sin(\omega_1 t + \phi_1) \\ -A_2\omega_2\sin(\omega_2 t + \phi_2) \\ \cdot \\ \cdot \\ \cdot \\ -A_n\omega_n\sin(\omega_n t + \phi_n) \end{bmatrix} \succ \ddot{\underline{\varphi}}(t) = \begin{bmatrix} -A_1\omega_1^2\cos(\omega_1 t + \phi_1) \\ -A_2\omega_2^2\cos(\omega_2 t + \phi_2) \\ \cdot \\ \cdot \\ \cdot \\ -A_n\omega_n^2\cos(\omega_n t + \phi_n) \end{bmatrix} \quad \textbf{(30)}$$

The modal frequencies are, ω_1 through ω_n. The amplitudes A_1 through A_n are redundant to the scaling of the mode shapes; thus the mode shapes may be normalized (scaled) as desired. Finally, the phase angles, ϕ_1 through ϕ_n, complete the solution to the modal vibrations. For the homogeneous initial value problem, all modes begin with the maximum amplitudes —derived from the initial conditions— implying ϕ_1 through ϕ_n equal zero. For the homogeneous impact problem (instantaneous velocity), all modes begin with zero amplitude and a phase of $\pm\pi/2$. (Note that the phase angle ϕ's are scaler values that are unrelated to the modal matrix, Φ, or its columns $\underline{\phi}_1$ through $\underline{\phi}_n$.)

The solution of (6), as formulated in (29) and (30) is found by solving the eigen equation. A convenient means to achieve this is to solve the homogeneous form of equation (6) one mode at a time. (Later it will be shown that the mode shapes of Φ are mutually orthogonal, thus a homogeneous solution of (6) will require each modal contribution to independently and simultaneously be homogeneous.)

$$\left.\begin{aligned} \boldsymbol{M}\underline{\phi}_i\ddot{\varphi}_i(t) + \boldsymbol{K}\underline{\phi}_i\varphi_i(t) &= \underline{0} \qquad (a) \\ \left(-\omega_i^2\boldsymbol{M} + \boldsymbol{K}\right)\underline{\phi}_i\varphi_i(t) &= \underline{0} \qquad (b) \\ \boldsymbol{M}^{-1}\boldsymbol{K}\,\underline{\phi}_i &= -\omega_i^2\underline{\phi}_i \qquad (c) \end{aligned}\right\} \quad \forall\, i \in \{1,2,\ldots n\} \qquad \textbf{(31)}$$

For non-trivial solution, the mode shape, $\underline{\phi}_i$, must not be composed entirely of zeros. Thus the modal frequencies are solved from the characteristic equation that would drive the determinate of (31b), $\det(-\omega_i^2 M + K)$, to zero. This will yield the n, natural frequencies. Once these are known, the respective non-trivial eigenvectors can be solved from (31) by substituting in each ω_i, to form n algebraic equations, and finding a solution for each element of $\underline{\phi}_i$. Note that as stated earlier, the scale of $\underline{\phi}_i$ is a free parameter, as a matter of convenience, the first non-zero element may be set to one. Then the remaining elements are completely determinate. A more convenient normalization will be developed in section 3.1.3.

Note that for the assumed solution form of (30) to be valid, the modal frequencies must be real valued. If any of the roots of (31) imply that ω_i has any imaginary component, then the assumed form is invalid.

Generally, this event will imply instability that will prevent an oscillating solution. This instability is avoided if both of the matrices M and K are positive semi-definite. (A sufficient condition for a real symmetric matrix to be positive semi-definite is for none of its eigenvalues to be negative. [21]) Further note for the formulation of (31c) that the inertial matrix is presumed invertible. This is not required for the eigen equation of (31b), but it is always true for the finite element beam formulation, and it is a convenient algebraic form to work with. (Note that if the stiffness matrix is invertible, $K^{-1}M\Phi_i = -\omega_i^{-2}\Phi_i$, is also a valid form of the equation.)

3.1.1 EIGENVECTORS OF REPEATED EIGENVALUES

The eigen problem of (31) becomes more challenging if their are repeated fundamental frequencies. If the j^{th} frequency is repeated k times this would imply that ω_j^k is a component of the characteristic equation of (31). The first eigenvector can be found as for non-repeated solutions. However, the remaining k-1 solutions can not be found as directly, but orthogonal eigenvectors may be constructed as follows: [6, 7, 14, 15]

$$\begin{aligned} K\Phi_j &= \omega_j^2 M\Phi_j \\ K\Phi_{(j+1)} &= \omega_j^2 M\Phi_{(j+1)} + M\Phi_j \\ &\vdots \\ K\Phi_{(j+k-1)} &= \omega_j^2 M\Phi_{(j+k-1)} + M\Phi_{(j+k-2)} \end{aligned} \qquad (32)$$

3.1.2 ORTHOGONALITY OF THE EIGENVECTORS

The orthogonality of the eigenvectors is the key that mathematically allows the dynamic problem, formulated using the finite element approach, to be broken down into a frequency domain perspective. As shown in (29) the dynamic problem has been decomposed into a finite number of spatial modes, directly coupled to a temporal component. It has been shown that the temporal solution is of the form of (30). Thus, the homogeneous response is a composed of a finite number of oscillating temporal frequencies. The orthogonality of the eigenvectors completes the decoupling of the problem by demonstrating that the contribution of each frequency is independent of the other frequencies. Thus, the eigenvectors form an orthogonal basis for the generalized coordinates.

To demonstrate the orthogonality of the eigenvectors of (31), provided that M and K are symmetric, consider any two solutions, and multiply them by the transpose of the other eigenvector: [14]

$$\begin{aligned} \omega_i^2 M\Phi_i &= K\Phi_i \quad (a) & \omega_j^2 M\Phi_j &= K\Phi_j \quad (b) \\ \omega_i^2 \Phi_j^T M\Phi_i &= \Phi_j^T K\Phi_i \quad (c) & \omega_j^2 \Phi_i^T M\Phi_j &= \Phi_i^T K\Phi_j \quad (d) \end{aligned} \qquad (33)$$

Now, transpose (33d), and note the symmetry of M and K:

$$\begin{aligned} \left(\omega_j^2 \Phi_i^T M \Phi_j\right)^T &= \left(\Phi_i^T K \Phi_j\right)^T \quad (a) \\ \omega_j^2 \Phi_j^T M^T \Phi_i &= \Phi_j^T K^T \Phi_i \quad (b) \\ \omega_j^2 \Phi_j^T M \Phi_i &= \Phi_j^T K \Phi_i \quad (c) \end{aligned} \tag{34}$$

Finally, subtract (34c) from (33c):

$$\left(\omega_i^2 - \omega_j^2\right) \Phi_j^T M \Phi_i = 0 \tag{35}$$

From (35) it is clear that: for $I \neq j$, $\phi_j^T M \phi_i = 0$. This is a definition of orthogonality. Since M is both invertible and positive semi-definite it is positive definite (all roots greater than zero). [21] Also, since ϕ_i is non-zero $\forall I \in \{1, 2, ...n\}$, $\phi_j^T M \phi_i \neq 0$. This relationship is critical in the basis transform that follows in section 3.1.4.

3.1.3 MASS NORMALIZATION OF THE EIGENVECTORS

As stated, the scale of the eigenvectors is a free parameter. In the derivation of the eigenvectors in section 3.1, they were normalized such that the first non-zero entry was set to one for algebraic simplicity. At this point, it will prove convenient to normalize the vectors such that the orthogonality relationship of (33 c,d) takes on a particularly useful form. (For convenience, we will subscript the mass normalized vectors with an N. This subscript will later be dropped, as mass normalization will be assumed for the remainder of this report.)

The desired orthogonality relationship is: [14, 15]

$$\begin{aligned} \underline{\Phi}_{N\,j}^T M \underline{\Phi}_{N\,i} &= \delta_{i,j} \\ \textit{Where} \quad \delta_{i,j} &= \begin{cases} 1 \ \textit{if} \ i = j \\ 0 \ \textit{if} \ i \neq j \end{cases} \end{aligned} \tag{36}$$

The orthogonality has already assured the zero value for $I \neq j$. What remains is to divide the un-normalized eigenvectors by the square root of the scaler product of $\phi_i^T M \phi_i$.

$$\begin{aligned} \Phi_i^T M \Phi_i &= m_i \\ \left(\frac{\Phi_i}{\sqrt{m_i}}\right)^T M \left(\frac{\Phi_i}{\sqrt{m_i}}\right) &= 1 \end{aligned} \tag{37}$$

Thus, the mass normalized eigenvector can be obtained from any scale by the following relationship:

$$\underline{\Phi_N}^i = \frac{\Phi_i}{\sqrt{\Phi_i^T M \Phi_i}} \tag{38}$$

Mass normalized eigenvectors will be assumed for the remainder of this report, unless otherwise stated, therefore the subscripted N, will be dropped.

3.1.4 MODAL TRANSFORMATION

The result of generating n, orthogonal, mass normalized, eigenvectors has been the development of the most simplified basis to represent the dynamics of (6). [22] A change of basis, or linear transformation may be accomplished that maintains the same system dynamics from the eigen perspective. This is termed modal transformation. This may be achieve by collecting the mass normalized eigenvectors into the modal matrix, Φ, of equation (29): [14, 15, 19]

$$\begin{aligned} M\Phi\ddot{\varphi}(t) + K\Phi\varphi(t) &= f(t) \qquad (a) \\ \Phi^T M\Phi\ddot{\varphi}(t) + \Phi^T K\Phi\varphi(t) &= \Phi^T f(t) \qquad (b) \\ I\ddot{\varphi}(t) + \Lambda\varphi(t) &= \Phi^T f(t) \qquad (c) \end{aligned} \tag{39}$$

Two major simplifications occurred in (39). First, the mass normalization transformed the mass matrix into the identity matrix, I. Second, as can be seen from (34b), the stiffness matrix, K, has been transformed into a diagonal matrix of squared fundamental frequencies. In general, the modal matrix is sorted so that the resulting modally transformed stiffness matrix, Λ, is composed of the squared frequencies in order of increasing frequency down the diagonal.

The result of equation (39c) is that the complicated solution of (6) has been transformed into a series of decoupled first-order differential equations. The vector, $\varphi(t)$, is called the modal coordinate vector. [19]

3.2 DAMPED SECOND-ORDER SYMMETRIC EIGENVALUES AND EIGENVECTORS

The homogeneous damped vibration problem of (19) can also be examined in from the eigen perspective, provided M, C_D, and K are symmetric, as follows. [23] First, in a similar approach to equations (26) and (27), convert the second order equations to first order form without inverting the inertial matrix while maintaining symmetry by implementing the trivial equality of (40):

$$M\dot{q} - M\dot{q} = \underline{0} \tag{40}$$

$$\tilde{A}\dot{\mathfrak{y}} + \tilde{B}\mathfrak{y} = \underline{0} \qquad (a)$$

Where:

$$\mathfrak{y} = \begin{bmatrix} \dot{q} \\ q \end{bmatrix} \quad (b) \quad \tilde{A} = \begin{bmatrix} 0 & M \\ M & C_D \end{bmatrix} \quad (c) \quad \tilde{B} = \begin{bmatrix} -M & 0 \\ 0 & K \end{bmatrix} \quad (d) \tag{41}$$

Note that the state space forms, (26) and (41), are different realizations of the same linear operator.

Assume a solution to (41) using the separation of variables similar to (29) of the form:

$$\mathfrak{y}(t) = \Xi\underline{\kappa} \qquad (a)$$

Where:

$$\Xi = \left[\underline{\xi}_1\ \underline{\xi}_2 \cdots \underline{\xi}_n\right] \qquad (b)$$

$$\underline{\kappa} = \left[\kappa_1 e^{\alpha_1 t}\ \kappa_2 e^{\alpha_2 t}\ \ldots\ \kappa_n e^{\alpha_n t}\right]^T \qquad (c) \tag{42}$$

The first time derivative of the exponential temporal terms result in:

$$\dot{\mathfrak{y}}(t) = \sum_{i=1}^{n} \alpha_i \underline{\xi}_i \kappa_i e^{\alpha_i t} \tag{43}$$

Substituting (42) and (43) back into (41) one mode at a time and noting the relationship between the upper and lower halves of $\underline{\xi}_i$ as a result of (41b) in particular:

$$\left.\begin{aligned} \tilde{A}\,\dot{\mathfrak{y}}_i + \tilde{B}\,\mathfrak{y}_i &= \underline{0} && (a) \\ \left(\alpha_i \tilde{A} + \tilde{B}\right)\underline{\xi}_i\,\kappa_i e^{\alpha_i t} &= \underline{0} && (b) \\ \begin{bmatrix} -M & \alpha_i M \\ \alpha_i M & \left(\alpha_i C_D + K\right) \end{bmatrix} \begin{bmatrix} \underline{\xi}_i{}^{Upper} \\ \underline{\xi}_i{}^{Lower} \end{bmatrix} &= \underline{0} && (c) \\ \underline{\xi}_i{}^{Upper} &= \alpha_i \underline{\xi}_i{}^{Lower} && (d) \end{aligned}\right\} \quad \forall\ i \in \{1,2,\ldots n\} \tag{44}$$

Recognizing that by including (44d) in (44c), and dropping the eigen index subscript, results in the form:

$$\left(\alpha^2 M + \alpha C_D + K \right) \xi^{Lower} = \underline{0} \tag{45}$$

Clearly, we may expect complex eigenvalues. Therefore, examining real and imaginary components of the eigen value, α_I and α_R of the characteristic equation, (45) without the eigenvector, separately:

$$\begin{aligned} \left(\alpha_R^2 - \alpha_I^2\right) M + \alpha_R C_D + K = 0 \qquad & (a) \\ -2\alpha_R \alpha_I M + \alpha_I C_D = \alpha_I \left(-2\alpha_R M + C_D\right) = 0 \qquad & (b) \end{aligned} \tag{46}$$

This demonstrates that the solution for one root, α, is also the solution of its complex conjugate, α^*, since the sign of the imaginary component factors out of both equations.

This relationship can be reiterated in the first order form of (44c), and further demonstrate that the eigenvectors of the complex conjugate eigenvalues are conjugate provided that the elements of A and B are real. (This provision guarantees that the complex conjugate of the product of a real matrix and a complex vector is equivalent to the product of a real matrix and the complex conjugate of the vector.)

$$\alpha_i A \xi_i + B \xi_i = \underline{0} \quad \succ \quad \begin{cases} \left(\alpha_i A \xi_i\right)^* + \left(B \xi_i\right)^* = \underline{0}^* \\ \alpha_i^* \left(A \xi_i\right)^* + B \xi_i^* = \underline{0} \\ \alpha_i^* A \xi_i^* + B \xi_i^* = \underline{0} \end{cases} \tag{47}$$

From this development, we can expect the first-order eigen solutions to contain twice as many eigenvalues as the original second-order form, but we anticipate that they will come in complex conjugate pairs, as demonstrated by (46) and (47).

The relationships between the eigenvectors is shown below for a sorted modal matrix, with subscripted Φ's in lieu of subscripted ξ^{Lower_i}s. (The Φ notation is consistent with the undamped eigenvectors of (29)):

$$\Xi = \begin{bmatrix} \alpha_1 \Phi_1 & \alpha_1^* \Phi_1^* & \alpha_2 \Phi_2 & \alpha_2^* \Phi_2^* & \cdots & \alpha_n \Phi_n & \alpha_n^* \Phi_n^* \\ \Phi_1 & \Phi_1^* & \Phi_2 & \Phi_2^* & \cdots & \Phi_n & \Phi_n^* \end{bmatrix} \tag{48}$$

3.3 FIRST-ORDER STATE-SPACE EIGENVALUES AND EIGENVECTORS

The first-order homogeneous state-space system of (25) through (28) can be related to the first-order state-space system of (41) as follows:

$$
\begin{aligned}
x &= \begin{bmatrix} 0 & I \\ I & 0 \end{bmatrix} \eta && (a) \\
\dot{\eta} &= -\tilde{A}^{-1}\,\tilde{B}\,\eta && (b) \\
\dot{x} &= -\begin{bmatrix} 0 & I \\ I & 0 \end{bmatrix} \tilde{A}^{-1}\tilde{B} \begin{bmatrix} 0 & I \\ I & 0 \end{bmatrix}^{-1} x && (c)
\end{aligned}
\qquad (49)
$$

$\tilde{A}^{-1}$ can be evaluated from its partitioned definition of (41b) using the Schur matrix inversion formula. [24] Also note that the inverse of the permutation matrix (skewed identity matrix) is itself. The result of combining the four matrices of (49c) follow:

$$
-\begin{bmatrix} 0 & I \\ I & 0 \end{bmatrix} \begin{bmatrix} \left(-M^{-1}\,C_D\,M^{-1}\right) & M^{-1} \\ M^{-1} & 0 \end{bmatrix} \begin{bmatrix} -M & 0 \\ 0 & K \end{bmatrix} \begin{bmatrix} 0 & I \\ I & 0 \end{bmatrix} = \begin{bmatrix} 0 & I \\ \left(M^{-1}K\right) & \left(M^{-1}C_D\right) \end{bmatrix} \qquad (50)
$$

As expected, this result is identical to the state-space formulation of (28a). As a result of this equivalence, the eigenvectors of the first-order state-space formulation of (25) through (28) are the same as Ξ of (37) with the bottom half partition and top half partition swapped as indicated by the simple transformation of (49a).

3.4 RIGID BODY MODES

Rigid body (degenerate) modes of vibration occur when a system is not fully constrained to an inertial reference frame. [14, 15, 18] In the case of transverse beam dynamics, two rigid body modes are possible: rotation and translation. The system matrices of the finite element formulation (8) and (9) are an example of an unconstrained structure. This implies that the system stiffness matrix is singular. Insight can be gained into the cause of this by investigating the singularity of the elemental stiffness matrix formulation of (3), and examining its implications on (9).

3.4.1 THE SINGULARITY OF THE FINITE ELEMENT STIFFNESS MATRIX

Close inspection of the second spatial derivatives of (1) used in (3) reveals that the third function is negative of the first. Further inspection reveals a relationship between all but three of the multiplied pairs of polynomials and the elemental length, h, prior to the integration. (The demonstration of this is included in the m-file, <rank_ki.m>.) Defining:

$$\alpha_i = \int_0^{h_i} E_i(x) I_i(x) \phi_1''(x) \phi_1''(x)\, dx = \int_0^{h_i} E_i(x) I_i(x) \left(\left(\frac{144}{h_i^6} \right) x^2 - \left(\frac{144}{h_i^5} \right) x + \left(\frac{36}{h_i^4} \right) \right) dx \qquad (a)$$

$$\beta_i = \int_0^{h_i} E_i(x) I_i(x) \phi_1''(x) \phi_2''(x)\, dx = \int_0^{h_i} E_i(x) I_i(x) \left(\left(\frac{72}{h_i^5} \right) x^2 - \left(\frac{84}{h_i^4} \right) x + \left(\frac{24}{h_i^3} \right) \right) dx \qquad (b) \qquad \textbf{(51)}$$

$$\gamma_i = \int_0^{h_i} E_i(x) I_i(x) \phi_2''(x) \phi_2''(x)\, dx = \int_0^{h_i} E_i(x) I_i(x) \left(\left(\frac{36}{h_i^4} \right) x^2 - \left(\frac{48}{h_i^3} \right) x + \left(\frac{16}{h_i^2} \right) \right) dx \qquad (c)$$

The elemental stiffness matrix (3) of an element, I, can be simplified to:

$$K^i = \begin{bmatrix} \alpha_i & \beta_i & -\alpha_i & (\alpha_i h_i - \beta_i) \\ \beta_i & \gamma_i & -\beta_i & (\beta_i h_i - \gamma_i) \\ -\alpha_i & -\beta_i & \alpha_i & -(\alpha_i h_i - \beta_i) \\ (\alpha_i h_i - \beta_i) & (\beta_i h_i - \gamma_i) & -(\alpha_i h_i - \beta_i) & (\alpha_i h_i^2 - 2\beta_i h_i + \gamma_i) \end{bmatrix} \qquad \textbf{(52)}$$

In this simplified form it is clear that the third row is the negative of the first, and the forth row is the first minus the second (or the negative of the third minus the second). The same is also true of the columns due to the symmetry of the matrix. Thus rank(K^i) is two and we can expect zero eigenvalues that result in rigid body modes of the single element. We can also notice, that by adding positive stiffness values to the main diagonal we may constraint the element. Clearly, at least two stiffness will have to be added to raise the rank by two, which raises the question: Will any two stiffness suffice? The intuitive answer is, of course, no. At least one stiffness will have to lateral. (Recall that the first and third rows and columns relate to lateral motion while the second and forth relate to rotational motion.)

To demonstrate that two rotational stiffness do not impose full rank on K^i, add a constant to the diagonal elements of rows two and four. Notice that row three is still the negative of row one; thus the element is not fully constrained. The lateral modes are not coupled with the rotational modes.

The converse is true. Two lateral stiffness do constrain the element. In fact, one lateral stiffness, combined with any other stiffness, will achieve full constraint. This is due to the dependence of the second rotational mode (the fourth row) on the lateral modes, prior to external constraint. (Note that we have arbitrarily decided to define the fourth row in terms of the first and second. We could also have defined the second in terms of the first and the fourth. Thus, it is not just the fourth row that is dependent on the lateral modes, both rotational modes may be formulated as a function of the other rotational mode, and either lateral mode.)

The lack of full rank of each elemental matrix results in a lack of full rank for the system stiffness of the finite element beam model. Because of the cascading combination of elemental matrices in (9), the rank of the full system matrix remain deficient by two. (For nn nodes, we combine (nn-1) elemental matrices, each of rank two. Thus the rank of the full system matrix, size 2×nn, is 2×nn-2.)

An analogous interdependence between rows and columns of the system stiffness matrix, K, occurs that allows external stiffness constraints at interior nodes to successfully constrain the integrated finite element model. In analogy with (9):

$$K = \begin{bmatrix} \alpha_1 & \beta_1 & -\alpha_1 & (\alpha_1 h_1 - \beta_1) & 0 & 0 \\ \beta_1 & \gamma_1 & -\beta_1 & (\beta_1 h_1 - \gamma_1) & 0 & 0 \\ -\alpha_1 & -\beta_1 & [\alpha_1 + \alpha_2] & [-(\alpha_1 h_1 - \beta_1) + \beta_2] & -\alpha_2 & (\alpha_2 h_2 - \beta_2) \\ (\alpha_1 h_1 - \beta_1) & (\beta_1 h_1 - \gamma_1) & [-(\alpha_1 h_1 - \beta_1) + \beta_2] & [(\alpha_1 h_1^2 - 2\beta_1 h_1 + \gamma_1) + \gamma_2] & -\beta_2 & (\beta_2 h_2 - \gamma_2) \\ 0 & 0 & -\alpha_2 & -\beta_2 & \alpha_2 & -(\alpha_2 h_2 - \beta_2) \\ 0 & 0 & (\alpha_2 h_2 - \beta_2) & (\beta_2 h_2 - \gamma_2) & -(\alpha_2 h_2 - \beta_2) & (\alpha_2 h_2^2 - 2\beta_2 h_2 + \gamma_2) \end{bmatrix} \quad \textbf{(53)}$$

Note that row five of (53) is a linear combination of the first and third rows. Thus, the addition of an external stiffness on the diagonal of row five, will in turn break-up the interdependence between the first and third rows. In general, the interdependence between rows cascades down the system matrix, and external constraints break-up the chains of interdependence. The independence of the lateral modes from the rotation modes persists. The odd rows are only interdependent with other odd rows.

3.5 MATLAB® REALIZATION OF EIGEN ANALYSIS

3.5.1 UNDAMPED MODE SHAPES AND FREQUENCIES

The file <eigen_2o.m> was written to compute the mode shapes and frequencies of both undamped, equation (6), and damped, equation (20), second-order equations of motion. In either case, the input matrices are first checked to be sure that they are square and of compatible size. The undamped eigenvalues and eigenvectors are then computed using equation (31b) and the generalized eigenvalue problem variation of MATLAB®'s <eig> command [12] in a straight forward manner. The circular frequency (radians per second) is then computed as the square root of the eigenvalues. Real frequencies

are guaranteed, provided that the mass and stiffness matrices are real valued, positive definite, and symmetric as was discussed in section 3.1. (Rigid body modes, corresponding to zero valued eigenvalues, sometimes result in a small imaginary content for the resulting zero frequency mode. This is presumed to be harmless numerical imprecision, but it does trigger warning messages.) Issues regarding repeated eigenvalues and rigid body modes are resolved by the MATLAB® <eig> command. [12]

3.5.2 DAMPED MODE SHAPES AND FREQUENCIES

Damped mode shapes and frequencies are computed if the optional damping matrix is included as an input variable to the file. Numeric accuracy limitations require the use of an imposed cut-off frequency to discriminate under-damped vibratory modes and rigid body modes. (Note that over-damped modes may exist.) The cut-off frequency chosen was five percent of the fundamental mode of the undamped system, which is computed first. This value may easily be changed if the need presents itself.

If the damping matrix was included, the conversion to the first-order form of equation (41) precedes the call to MATLAB®'s <eig> command [12] to implement equation (44b). Extraction of the mode frequencies is accomplished by taking the absolute value of the imaginary component of each pair of complex conjugate eigenvalues. The unit of the frequency is radians per second, as can be seen from equation (42c) in light of Euler's equation (polar form [21]):

$$e^{(\alpha_R + \alpha_i i)t} = e^{\alpha_R t}\left(\cos(\alpha_i t) + i\sin(\alpha_i t)\right) \qquad \textbf{(54)}$$

Note also from Euler's equation that since the eigenvalues that contain imaginary components come in conjugate pairs —due to the symmetry of the system matrices— the imaginary content of the time domain response cancels out: $I\sin(\alpha_i) + I\sin(-\alpha_i) = 0$.

Extraction of the second-order damped mode-shapes from the first-order eigenvectors is achieved by identifying the eigenvalues with positive imaginary content. (Thus neglecting the corresponding negative conjugate.) The mode-shapes are extracted by taking the real part of the upper half of the eigenvectors that correspond to the identified eigenvalues. Note that the first-order eigenvectors identified by MATLAB® are normalized to a Euclidian length of one. Since the eigenvectors include imaginary content, MATLAB® must, in general, multiply each eigenvector by a complex scaling coefficient. This may mask the underlying structure identified in (48). The structure of (48) may be partially recovered by identifying a new complex coefficient for each independent eigenvector that cancels the imaginary content of the lower half of (48). Further, the columns corresponding to conjugate eigenvectors may be normalized by a real-valued scaler value to obtain a one-to-one match between the lower halves of conjugate eigenvectors. Once these steps are taken, the relationships indicated by (44d) and (47) are revealed.

3.5.3 IDENTIFICATION OF RIGID BODY MODES

Rigid body modes correspond to eigenvalues of zero in either case of damped or undamped systems. MATLAB® 's numerical approximation of the damped eigenvalues makes the identification of the zero eigenvalues more challenging. The cut-off frequency that was used to discriminate flexible modes from over damped modes is employed to identify eigenvalues that are *close enough* to the origin to be considered as rigid body modes. This identification is not robust; it is susceptible to errors. For this reason, the expected number of rigid body modes is computed from the rank deficiency of the stiffness matrix (see section 3.4.1). If a discrepancy is found between the computed and identified number of rigid body modes, a warning is written to the screen.

3.5.4 CYCLIC FREQUENCY

The cyclic frequency is obtained by dividing the circular frequency, radians per second, by $2\times\pi$ radians per cycle. The resultant frequency is in units of cycles per second or Hertz.

3.5.5 NORMALIZATION OF THE MODE-SHAPES

The undamped eigenvectors, or the damped mode-shapes extracted from the first-order eigenvectors, are mass normalized using the relationship of (38). In addition, a consistent sign convention is imposed that results in a positive initial rotation of the beam (left end pointing down). This reduces confusion that may be caused by the sign ambiguity of (38) that results in seemingly random orientations of the mode-shapes about the undeformed axial axis.

3.5.6 SORTING OF MODES IN ORDER OF INCREASING FREQUENCY

The cyclic frequencies are sorted in order of increasing frequency. The columns of the mode-shape matrix are rearranged to maintain a one-to-one correspondence with the sorted mode frequencies.

3.5.7 EVALUATION OF ORTHONORMALITY OF THE MODE-SHAPES

The orthogonality and mass normalization of the mode-shapes imposed by (38) is validated by computing the norm of the following matrix:

$$\mathbf{\Phi^T M \Phi - I \approx 0} \qquad \textbf{(55)}$$

As indicated by (55), the result would ideally be a matrix the same size as the mass matrix and full of zeros. Thus, a small value was selected as a criteria to compare against the resulting norm to validate, or call into question, the orthonormality of the mode-shapes. If the test is failed, a warning is written to the screen.

This test has revealed that MATLAB® does not generally produce orthogonal mode-shapes corresponding to repeated rigid body modes (as one could expect). For the other cases, one rigid body mode or a fully constrained system, MATLAB® produces nearly orthogonal mode-shapes in most cases.

3.5.8 PLOTTING OF BEAM DEFORMATION AND MODE-SHAPES

An m-file, <beam_plot.m>, was written to plot the beam deformations as a function of the generalized coordinates. The input information required includes the node coordinates, the axial position vector, the generalized coordinate vector, a scaled geometry matrix, the constraint vector, and if an external vibration absorbers are included, an input vector of the generalized coordinates to which the absorbers are coupled. (Note that the mode-shapes evaluated above consist of a vector of generalized coordinates values.)

After several validation checks of the input arguments, the deflections of the centerline of each finite element of the beam are computed using (7). This requires the use of the file, <fem_interp.m>, discussed in section 2.2.4. The deflections of each element are combined into a total beam deflection vector, that maintains a one-to-one correspondence with the axial position vector.

Once the centerline deflection vector is computed, an exaggerated plot of the actual distorted beam shape may be obtained adding the centerline deflection to a scaled form of the geometry matrix that was computed when the input beam geometry was formulated. (See section 2.2.2.) The geometry matrix is scaled prior to calling the plotting routine, to achieve an aesthetically pleasing plot of the deformed beam geometry. (If the scaled geometry dimensions are much greater than the deflections of the center line, the deflection will be obscured. Conversely, if the scaled geometry dimensions are much smaller than the deflections, the geometry will be obscured resulting in a plot that approaches as narrow line as the difference is exaggerated.) For clarity the centerline is plotted with the finite element node locations explicitly tagged by x's.

External constraints are depicted by a thick line connecting the beam's generalize coordinate location to its undeformed location. Thus they may create the impression of an elastic member, tying the beam to the inertial reference frame at the nodal constraint location.

If an external absorber is coupled to the barrel, the axial position of the external center of mass is directly indicated by the location of its generalized coordinate. The position of the external mass is directly plotted as a character, * or o, using its lateral deflection explicitly. In addition, a line coupling the external mass to its beam generalized coordinate is included. It is important to note that provisions to *correctly* depict rotational absorbers has not been included. To do so would require explicit information on the plot aspect ratio and scaling, and could still lead to confusion due to the exaggerated slopes of the beam itself. Thus far, provisions to better represent them have not been warranted. For now, the rotational absorbers are indicated by the, *, character, and lateral absorbers by the, o, character. The angular deflections of the rotational absorbers are simply plotted as lateral deflections that still convey the relative phase and motion of these coupled inertias.

4 MATLAB® DYNAMIC ANALYSIS CASE STUDIES

MATLAB® script m-files [12] were written to demonstrate some of the analysis that is possible with the collection of function files documented in this report to conduct dynamic analysis of non-uniform beams. It is the purpose of this section to conduct a step-by-step case studies of a nonuniform, partially constrained and fully constrained beams. In the process, a pole-zero map will be constructed, a Bode frequency response plot will be developed, a time domain simulation of the dynamic response of the beam to an input vector will be computed, and the modeling will be validated.

4.1 THE RIGID BODY XM291 CASE

This first case study section will examine the behavior of an XM291 gun system, that is not coupled to an elevation mechanism. Thus, it is only partially constrained by the trunnion mount and exhibits rigid body motion manifest in angular deflections. This model could be used as the open-loop plant for the formulation of an elevation feedback control.

4.1.1 PRELIMINARIES

The script file, <XM291rb.m>, that implements this first part of the case study is broken up into sections that execute commands to achieve one or more closely related modeling goals. At the beginning of each section, the variables to be defined in the section are indicated, along with the variables to be altered and used in the section. Further, a brief description of the section is provided. Interim variables that are defined within a section, but not used by later sections, are sometimes set to null values at the end of the section to maintain clarity, and keep the variable work-space manageable.

Prior to reading in the beam geometry to be studied, we have found it convenient to establish two flags to indicate whether or not plots should be displayed, and if so, whether or not they should be printed. For the remainder of this case study, it is assumed that the plot flag is on.

Finally, a plot label is set to automatically annotate several of the plots that are generated. In this case, the label is set to: XM291rb. The subject will be the analysis of the XM291, laterally constrained by a stiff spring approximation to a frictionless pair of trunnion bearings in the vertical plane. This will result in one rigid body rotational mode, and set the stage for analysis of the beam dynamics, subjected to force input at the elevation mechanism location.

4.1.2 BEAM GEOMETRY

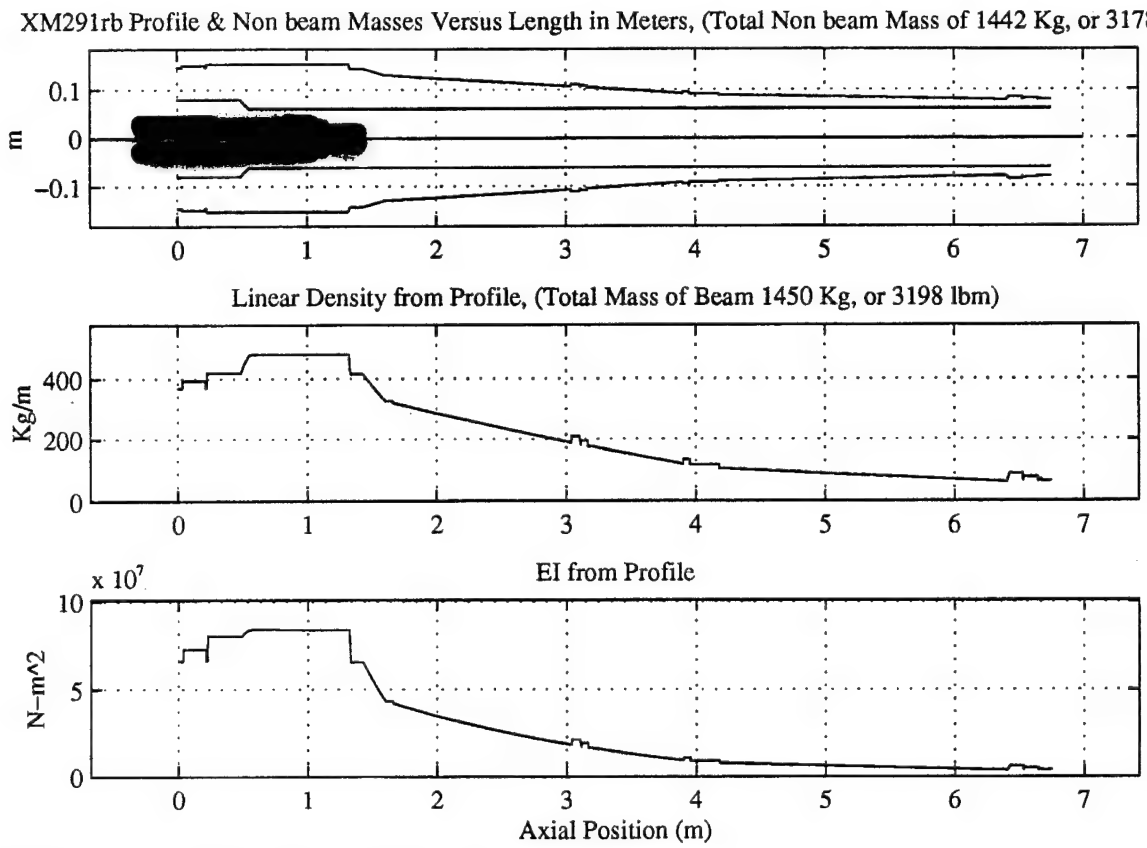


Figure 1 Output Plot of M-File, <geomf_XM291.m>.

The geometry of the XM291 gun barrel is read in using the function file, <geomf_XM291.m> as described in section 2.2.2. The plot of Figure 1 is automatically included by calling the file with the plot label as an input variable.

Some of the features of Figure 1 are worthy of comment. The top plot indicates the inner and outer radii reflected about the centerline. This is particularly useful for debugging the input geometry file, especially for a geometry file with as many detailed entries as the file for the XM291. Also notice in this top plot that the black region towards the rear of the barrel indicates the distribution of the extraneous mass, and that the title includes the total non-beam mass for validation. This mass is kept separate from the beam linear density depicted in the second plot with the total beam mass indicated in its title. By segregating the two densities, the non-beam masses do not obscure the validation of the beam density. (If the two densities were combined, the second plot would not lend as much qualitative or visual validation of the computed linear densities.) Finally, the third plot indicates the linear distribution of beam stiffness.

4.1.3 FINITE ELEMENT MESH GENERATION

The first requirement to generate the finite element mesh is to specify the locations where external constraints or forces are expected to interact with the beam. In the case at hand, the barrel is to be constrained at the trunnion location and forced from the elevation mechanism location. Therefore, these locations need to be finite element node locations and are included as imposed node locations for the meshing m-file, <fem_mesh.m>. In addition, the desired number of elements has been chosen to be seven. A relatively small number was selected to facilitate dynamic analysis, and to keep the system matrices and generalized coordinate vectors small enough for simple inspection and plot labeling. This information is combined with the four data vectors generated earlier results in the meshing of Figure 2.

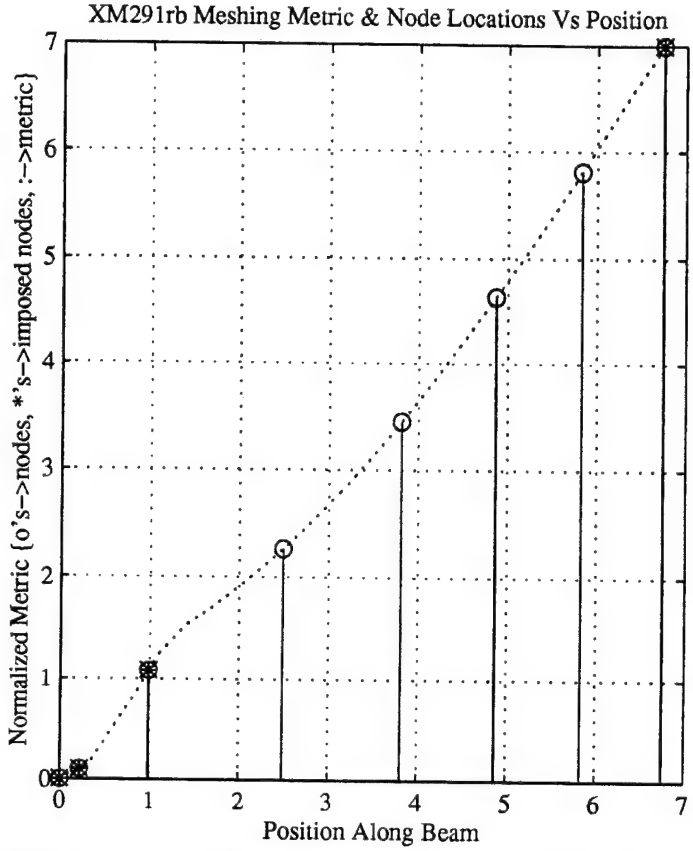


Figure 2 Output Plot of M-File, <fem_mesh.m>.

Figure 2 depicts the meshing metric that was developed in section 2.2.3. The purpose of the metric was to evenly divide the *super* elements, between the imposed node locations, into the allotted number of elements. The imposed node locations are indicated by a filled circle in the plot, while the free node locations are indicated by an empty circle. (Beam boundaries —by default— are imposed node locations.) In this rather simple example, only the *super* segment after the elevation mechanism is allotted any *free* elements. The five elements within the *super* element are then evenly spaced with respect to the metric.

4.1.4 COMPUTATION OF THE SYSTEM MATRICES

The mass and stiffness matrices of the free-free beam are computed from the raw data vectors, and the coordinates of the mesh nodes using the m-file, <fem_form.m> as described in section 2.2.5 The constraint of the trunnion bearings is modeled as a stiff lateral spring (750,000 lb_f/in) with no rotational resistance and a damping value equivalent to the Rayleigh stiffness proportional damping given below. The generalize coordinate to which the constraint is coupled is identified, and all three values are integrated into a one-by-three external constraint data matrix.

The free-free beam matrices are then modified to incorporate the lumped effects of the coupled rigid body masses discussed in sections 2.2.2 and 2.3.2. The Rayleigh damping is then computed for the modified system matrices using values of zero for α and 10^{-3} for β. Finally, the external constraint is added. This is done in the m-file, <fem_lump.m> as described in section 2.5. The final result is depicted in the images of the mass and stiffness matrices in Figure 3.

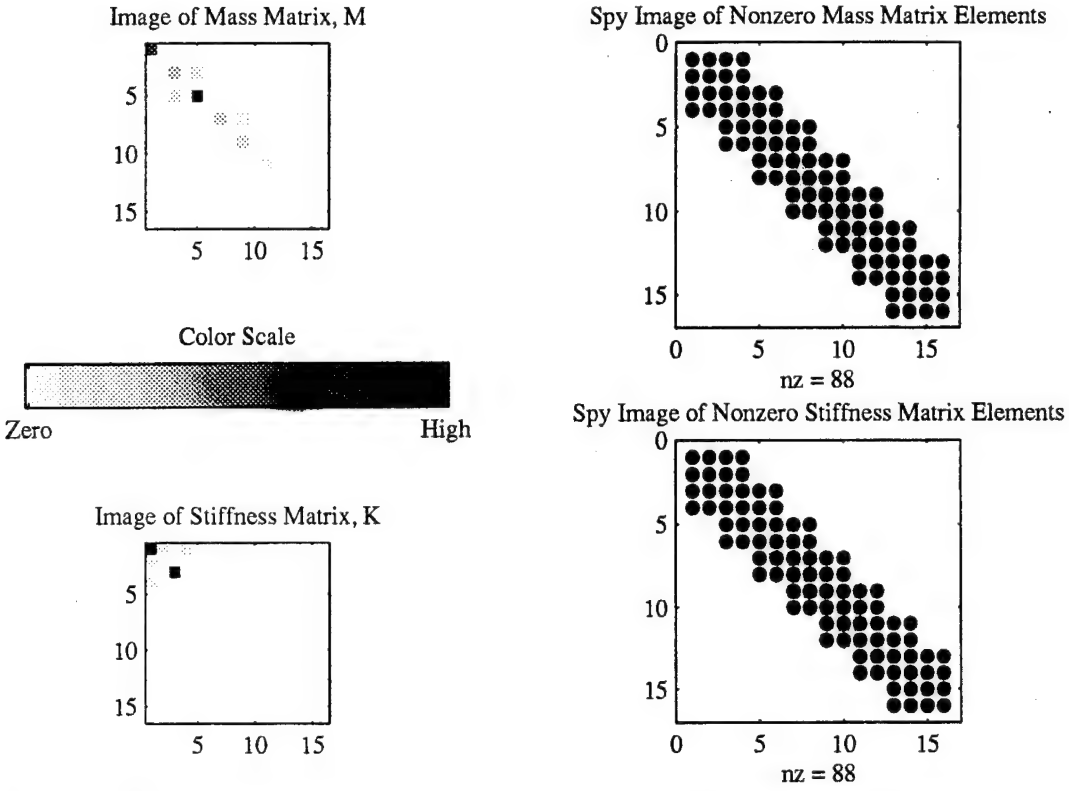


Figure 3 Image of System Matrices as Plotted by the M-File, <XM291rb.m>, Section 5.

Figure 3 depicts the image of the completed system mass and stiffness matrices. The shaded images indicate the locations of relatively large inertias and stiffness (near the breech). Since many non-uniform beams are tapered, and since inadvertent dimensional scaling of rotational versus lateral parameters occurs, the color scale of the images may obscure the distinction between zero elements, and *small* elements. To address this issue, the non-zero cells of the matrix, regardless of their magnitude, are illustrated to the right using MATLAB®'s <spy> command. [12] This verifies the cascading construction of both system matrices from four-by-four elemental matrices.

4.1.5 UNDAMPED EIGENVECTOR AND FREQUENCY DETERMINATION

The undamped second-order eigenvectors and frequencies are computed using the m-file, <eigen_2o.m>. This file conducts the eigen analysis as described in section 3.5.1. The resulting eigenvectors are then plotted using the m-file, <beam_plot.m>, as described in section 3.5.8. The results of the first six modes are depicted in Figure 4.

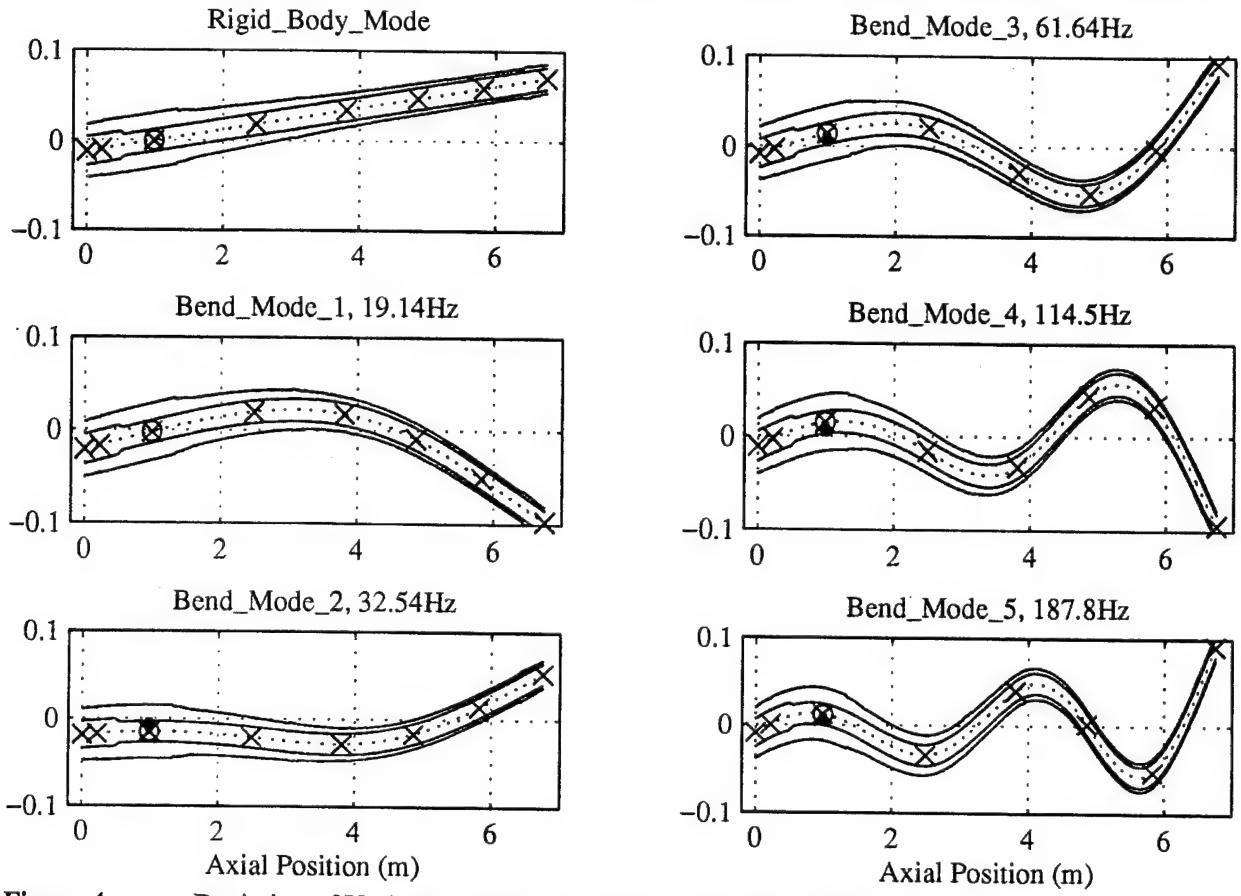


Figure 4 Depiction of Undamped Eigenvectors Computed by <XM291rb.m>, Section 6.

4.1.6 DAMPED MODE-SHAPE AND FREQUENCY DETERMINATION

The damped mode-shapes and frequencies are also computed using the m-file, <eigen_2o.m>. This file conducts the damped eigen analysis with the inclusion of the damping matrix as an input variable. The results of the first six bending modes are depicted in Figure 5. (The rigid body mode is not redisplayed, although it is computed.) The last mode-shape appears to be out of order. Insight as to the cause of this will be made later in section 4.1.8. Note that it is a good practice to conduct the same analysis with more elements to examine if the mode shapes, within the frequency band of interest, change in character. Finite element modeling of higher modes improves with more elements (as discussed in section 2.1.1 and demonstrated in section 4.4.2) thus providing a good check of reduced element models.

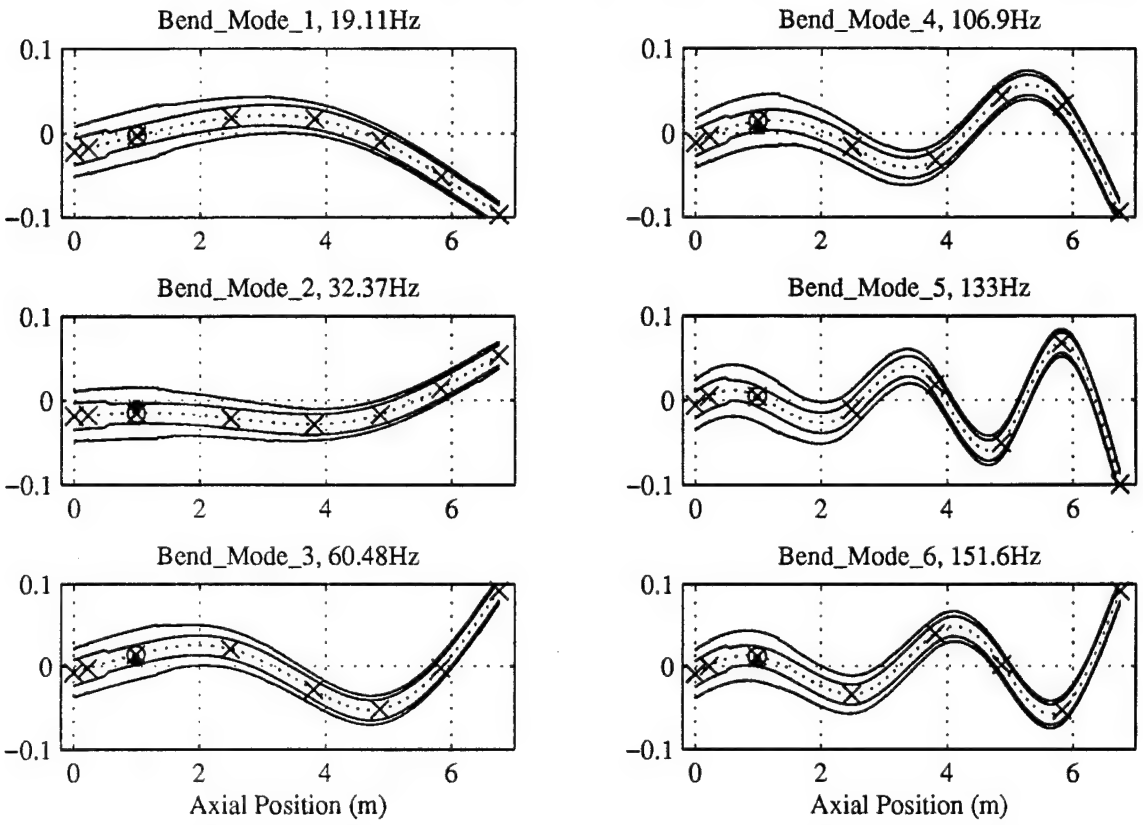


Figure 5 Depiction of Damped Mode Shapes Computed by <XM291rb.m>, Section 7.

4.1.7 CONVERSION TO FIRST-ORDER STATE-SPACE

Conversion of the second-order equations of motion to the first-order state-space form enables the use of MATLAB®'s library of dynamic analysis tools. The conversion is executed by the m-file, <fem2ss.m>, which implements equation (28) for the state-space equations of motion of equation (27). The resulting matrices contain some zero and identity sub-matrices that are revealed in figure 6, which was generated using the <spy> command. [12]

Once converted to state-space, the inclusion of irrelevant input and output variables may be removed using MATLAB®'s <ssselect> command. [25] It is particularly helpful to reduce the multi-input, multi-output (MIMO) system to a single input, single output (SISO) system. In the case of gun dynamics, a relevant choice for a SISO system would be the elevation mechanism as the sole input, and the muzzle pointing angle as the sole output. This is executed in section eight of the m-file, <XM291rb.m>.

4.1.8 POLE-ZERO MAP

Once the first-order matrix representation of the system is in the MATLAB® work-space, a pole-zero map may be constructed to provide graphical insight into the behavior of the system using MATLAB®'s <pzmap> command. [25]

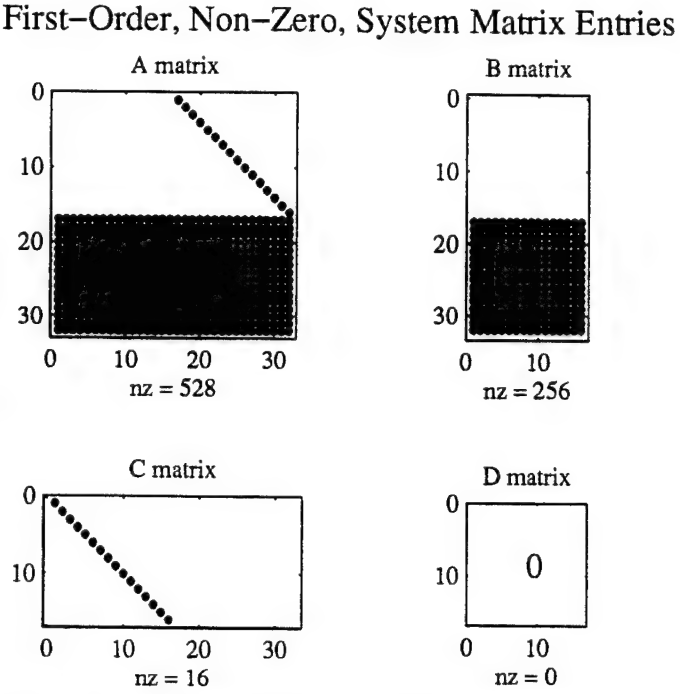


Figure 6 First-Order System Matrix Population Computed by <XM291rb.m>, Section 8.

Figure 7 depicts a region of the pole-zero map of the system. The poles are indicated by x's, and the zeros by o's. MATLAB®'s <sgrid> command [25] was used to generate the lines of constant natural frequency, and critical damping ratios in increments of twenty percent. (The zero damping ratio is collocated with the positive and negative imaginary axes, while the unity ratio is collocated with the negative real axis.) The map reveals several important properties of the damped system. First, the zeros in the right hand plan demonstrate that system is a non-minimum phase system. [26, 27] Second, the effect of stiffness proportional damping is more pronounced in the higher frequencies as can be seen by the higher damping ratio's of the higher natural frequency modes. Third, flexible modes beyond the sixth undamped flexible mode are now over damped, and their poles have migrated along the negative real axis. Fourth, the single rigid body mode is represented by the pole on the origin. Fifth, the effect of the damping, is to slide the flexible poles down the constant radii of natural damping. This allows the correlation of damped mode-shapes to their undamped eigenvectors. (This explains the unusual sequence of damped mode-shapes in Figure 5.) Finally, notice that the flexible poles all lay on the locus of a circle, centered on the negative real axis at the point to which the zeros, and over damped poles seem to be migrating towards. This is very reminiscent of root locus construction and could provide valuable insight.

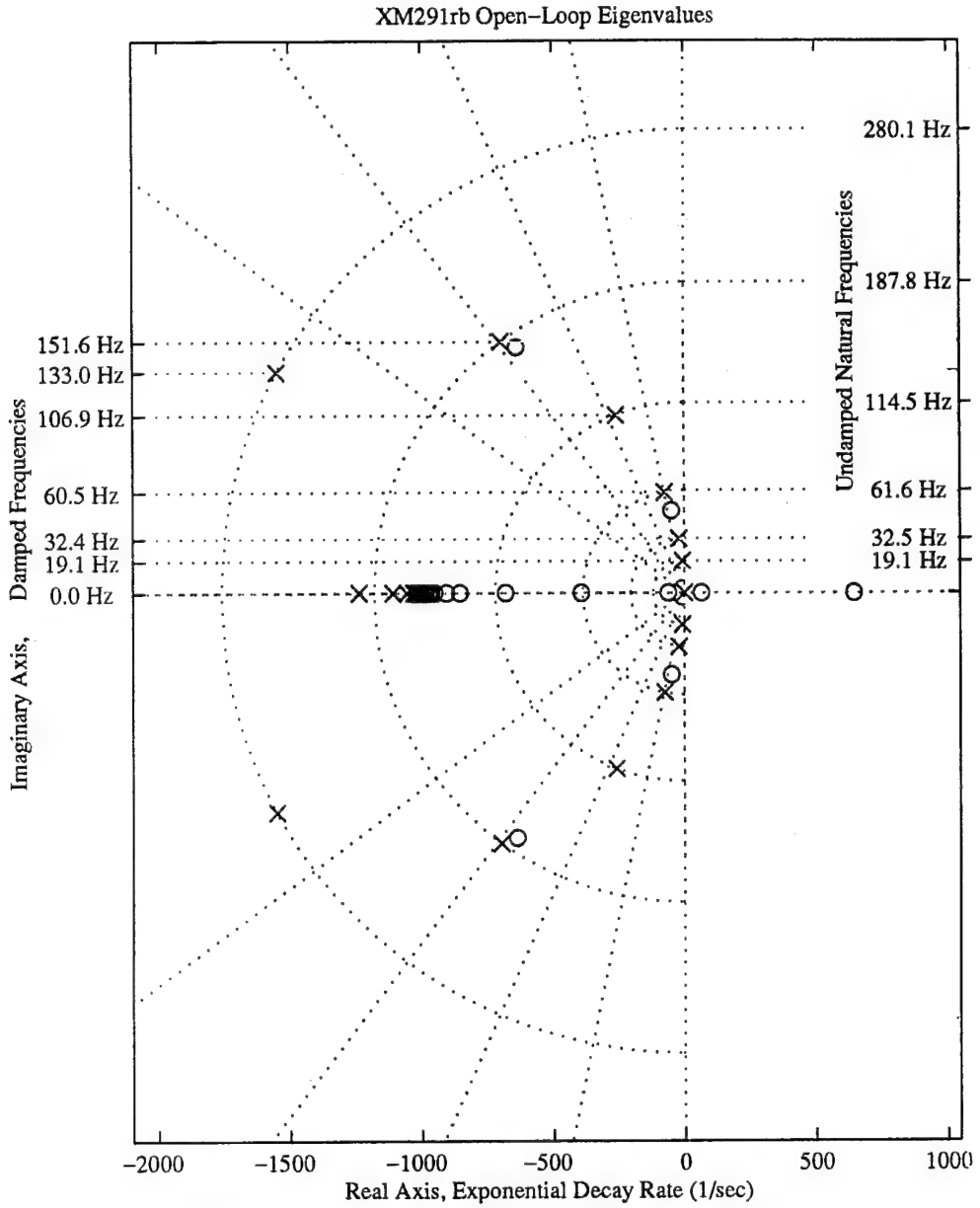


Figure 7 Pole-Zero Map Generated by the M-File, <XM291rb.m>, Section 9.

4.1.9 FREQUENCY RESPONSE BODE DIAGRAM

The frequency response of a dynamic system indicates the steady-state response, y(t), of the system to a sinusoidal input, u(t): [25]

$$\begin{aligned} u(t) &= A\sin(\omega t) \\ y(t) &= kA\sin(\omega t + \phi) \end{aligned} \qquad \textbf{(56)}$$

Classical control techniques relied heavily on these techniques because the Laplace domain was the best available means to conduct dynamic analysis prior to the introduction of state-space techniques in the nineteen-sixties. Frequency response analysis complements the state-space and eigen analysis by providing indications of stability, in addition to a quantitative and qualitative perspective.

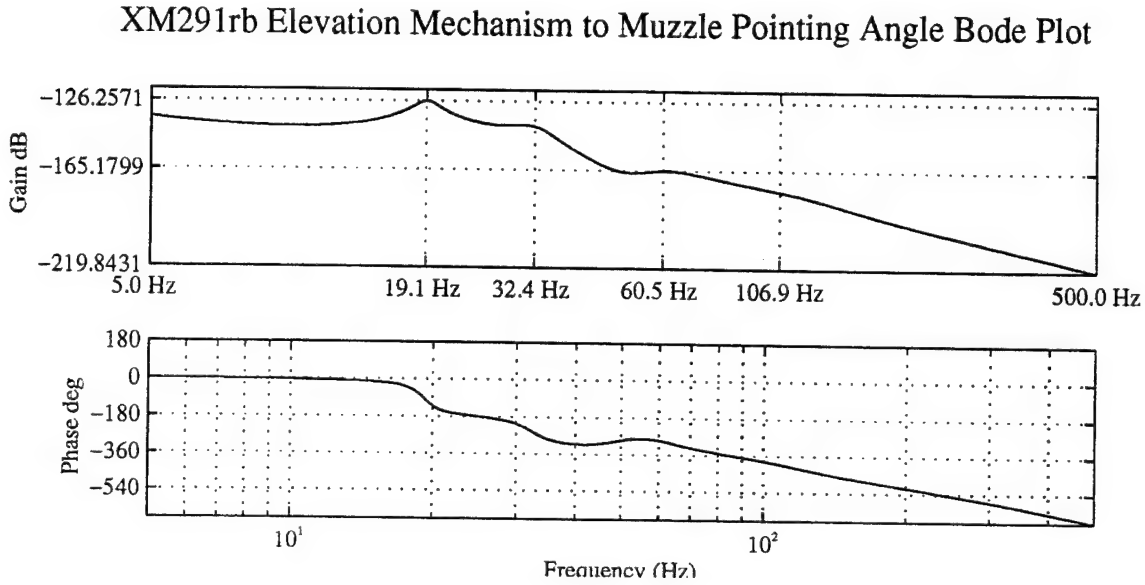


Figure 8 Bode Diagram Generated by the M-File, <XM291rb.m>, Section 10.
Figure 8 is a Bode Diagram of the SISO response of the muzzle pointing angle to the elevation mechanism. It consists of two plots, the upper relates the gain of the system, k in equation (56), to the excitation frequency, ω, and the lower plots the phase lag, ϕ, to the frequency. The gain is represented in decibels. (A decibel is related to the gain as: $dB = 20 \log_{10}(k)$.)

Examination of the upper plot indicates that the response is greatest at the first mode, followed by the second mode, and then tapers off to a very small response. This information may be used to argue that the system is most susceptible to disturbances in the frequency range near the first two modes. If any modifications are contemplated for the system, it would be highly desirable shift this frequency response band away from known disturbance sources. In the case of tank cannon, disturbances from terrain induced vibration are concentrated at the low end of the spectrum. (A 63,000 Kg M1A2 tank [28] with any viable suspension makes an impressive low-pass filter.) Therefore, it is very desirable to keep the fundamental modal frequencies as high as possible.

4.1.10 IMPULSE RESPONSE OF UNCONSTRAINED BARREL

The response of the muzzle pointing angle to a Newton Second impulse at the elevation mechanism may be obtained using MATLAB®'s <impulse> command [25]. Since this system contains a rigid body mode, the net motion of the barrel drifts away from the initial state while the flexible modes are vibrating about this drift. The important thing to notice in Figure 9, is the initial transient in the response before the gross motion of the lower frequency modes is established. This is the non-minimum phase effect discussed in section 4.1.8.

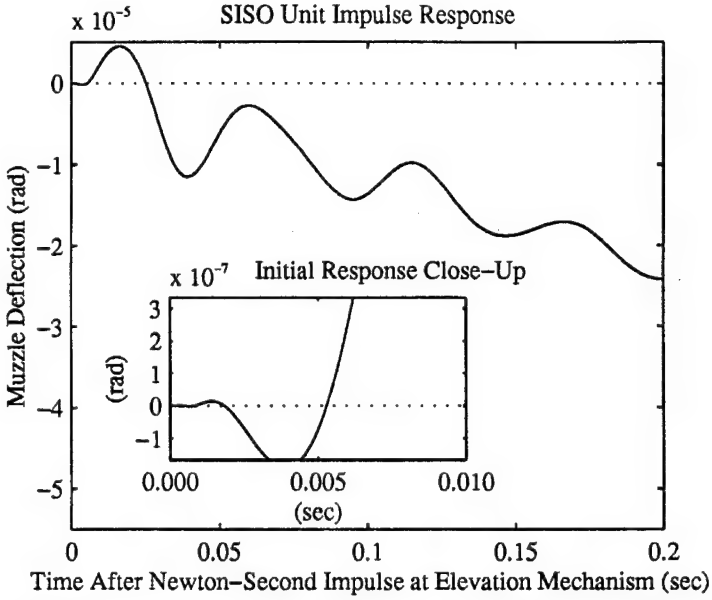


Figure 9 Impulse Response Generated by the M-File, XM291rb.m, Section 11.

4.2 FULLY CONSTRAINED XM291

It is important to realize that all of the analysis thus far has been conducted on a barrel, that is not fully constrained. This analysis lends itself to the design of the elevation mechanism feedback control, where the unconstrained barrel is the open-loop plant. In reality, the feedback controlled elevation mechanism is used to point the barrel to the desired elevation. In the lab, the barrel is mounted in a stand with a near rigid bar constraining the motion of the barrel. This laboratory bar, may be modeled as a stiff lateral spring with no rotational restoring force. (This model also emulates a high gain proportional feedback controller.)

Analysis of the fully constrained XM291 is executed by the m-file <XM291fc.m>. This file mimics the analysis of the file <XM291rb.m> discussed in section 4.1 until a second lateral constraint is added at the location of the elevation mechanism. The constraint value assigned to it is two-thirds the spring constant of the trunnion constraint. This added constraint eliminates the rigid body motion of the barrel, thus providing the opportunity to examine the response of the stable structure.

4.2.1 DAMPED MODE-SHAPE AND FREQUENCY DETERMINATION

The damped mode-shapes are again computed using the m-file <eigen_2o.m> in section 6 of the script file <XM291rb.m>. The first modes are depicted in figure 10. The change in the modes from figure 5 are worthy of a couple comments.

The first bending mode of the constrained model exhibits the classic, quarter wave cantilevered deflection. This demonstrates that the elastic constraints are quite stiff relative to the barrel, and have successfully introduced a new mode of vibration.

It is also interesting to note that the other modes appear to be only slight modifications of the unconstrained modes; with the differences becoming less pronounced at the higher frequencies.

4.2.2 BODE DIAGRAM

After the computation of the new finite element matrices, the first-order state space conversion is effected as in section 4.1.7. The frequency response of the new system is computed in section 8 of the m-file <XM291fc.m> and displayed in figure 11. It has been altered appreciably from the unconstrained case to reflect the introduction of the new —low frequency— mode. Unlike the Bode plot of figure 8, this plot was computed for input forcing at the trunnion. This provides a model of the gun subject to disturbances introduced at the trunnion, and *controlled* by the restoration force at the elevation mechanism. In this case, the restoration force merely models a linear spring.

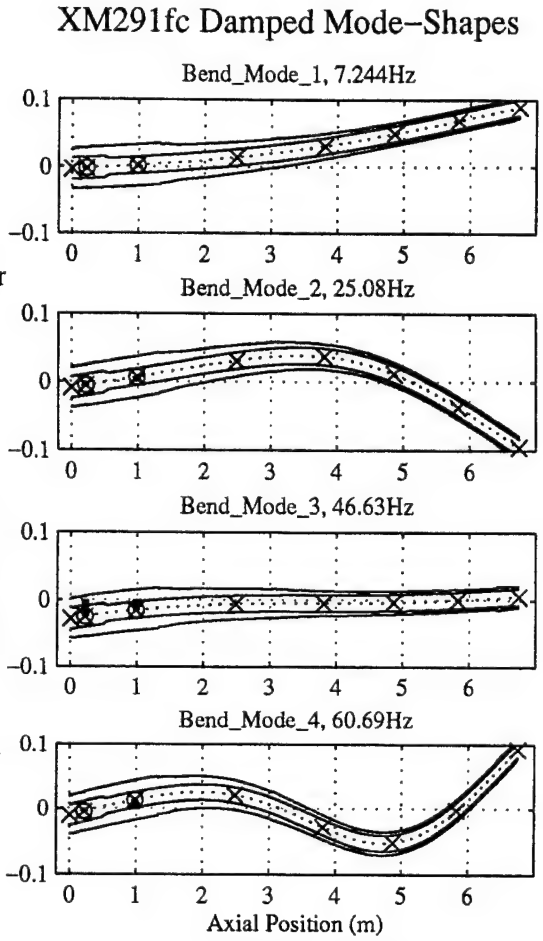


Figure 10 Depiction of Mode Shapes for Fully Constrained XM291.

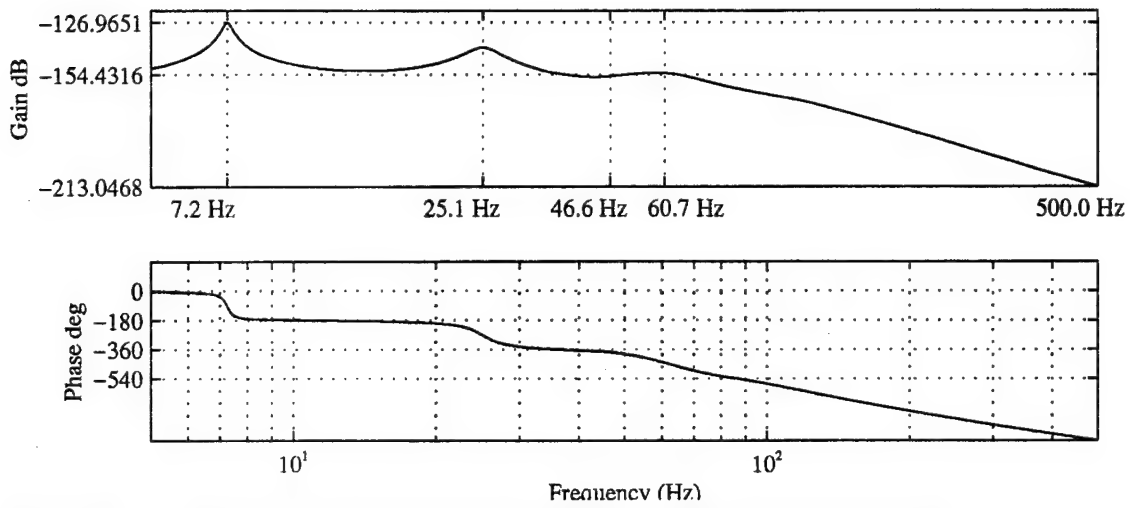


Figure 11 Bode Diagram Generated by the M-file, <XM291fc.m>, Section 8.

4.2.3 IMPULSE RESPONSE

The impulse response of the constrained barrel is depicted in figure 12. The lack of a rigid body mode prevents the response from drifting off to infinity as occurred in figure 9. Examination of figure 12 reveals that several modes, of different frequencies, are oscillating about the equilibrium point of zero. The higher frequencies, distinguishable by their short periods, clearly dissipate more quickly than the low frequency fundamental mode. The fundamental mode, near seven Hertz, dominates the response after a quarter of a second. This illustrates the effect of the stiffness proportional damping employed as discussed in sections 2.4 and 4.1.4; the higher frequencies are penalized more than the lower frequencies.

Also note that the non-minimum phase effect discussed in sections 4.1.8 and 4.1.10 is clearly present in the constrained barrel.

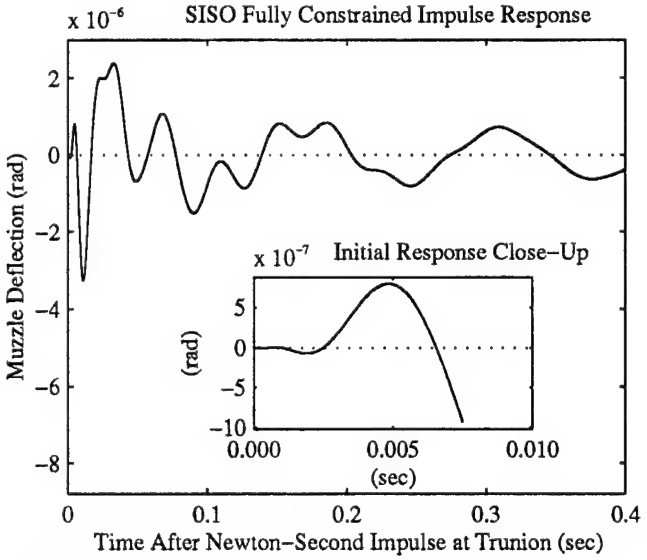


Figure 12 Impulse Response Generated by the M-file, <XM291fc.m>, Section 9.

4.2.4 STEP RESPONSE

Since the barrel is now fully constrained, a step response simulation may be executed as is done in section 10 of <XM291fc.m>. The step response simulates the effect of a sudden, but constant, application of a one Newton vertical force at the trunnion location. This is very similar to the effect of an impulse, with two major differences.

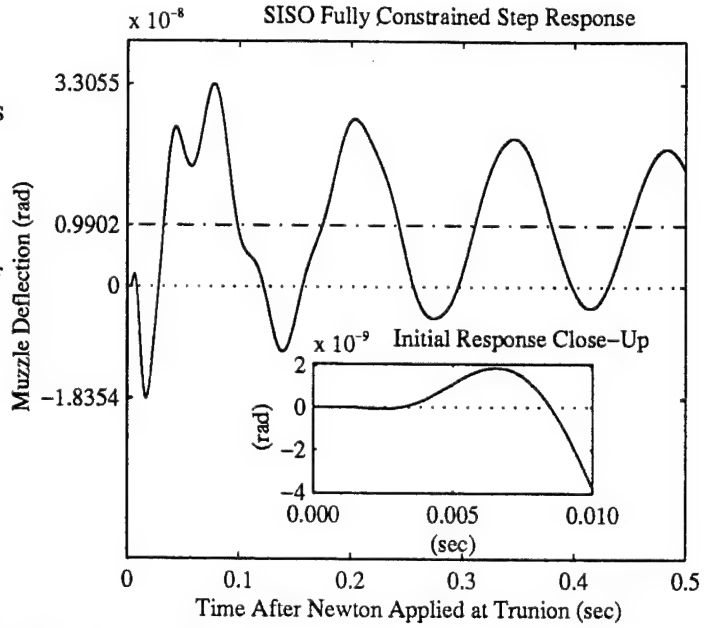


Figure 13 Step Response Generated by <XM291fc.m>, Section 10.

First, the step response results in a static offset. In the case of the impulse response of a fully constrained system, the system eventually returns to its equilibrium state which is the same as the initial state. However, the step response, invokes a constant loading that results in a new equilibrium state, that may be calculated via the final value theorem. [26, 27] This method was used to plot the dot-dashed line of the new equilibrium deflection. It was evaluated in section 10 of the m-file, <XM291fc.m>, by first converting the truncated state-space system to its transfer function equivalent using MATLAB®'s <ss2tf> command [25]. The Laplace "*s*" operator was then effectively set to zero by only taking the constant coefficients of the numerator and denominator. (This method does not solve for limits, so it will fail when the denominator coefficient is zero. A <while> loop [12] could be invoked to seek the first non-zero denominator coefficient, and then compute the limit more reliably.)

Second, relative to the impulse response of figure 12, the higher frequency content is low. The reason for this is that a time domain impulse results in a flat frequency response. (A cut off does occur in physical systems that is a function of the impact mass, and the effective resilience of the impactor. [29]) However, the frequency content of the step input is: [30]

$$\mathcal{F}\{ H(t) \} = \frac{1}{2}\delta(f) - \frac{i}{2\pi f} \quad \textit{Where:}$$

$\mathcal{F}$ *is the Fourier Transform*
H *is the Heaviside Unit Step Function*
δ *is the Dirac Impulse Function*
t *is the Time Domain Variable*
f *is the Frequency Domain Variable*
i *is* $\sqrt{-1}$

(57)

Clearly, the step input excites less response in the higher frequencies.

4.2.5 STATIC GRAVITY DEFLECTION

Gravity loading exerts a distributed force along the span of the XM291 barrel that is proportional to its linear density. In fact, the gravity force vector, to the resolution of the beam geometry, is the acceleration of gravity multiplied by the total linear density of the barrel and any extraneous mass that is attached to it (assuming that the undeformed barrel is horizontal). This input vector is computed in section 11 of <XM291fc.m>, and used by the m-file, <fem_force.m>, developed in sections 2.1.2 and 2.2.6. The result is a system force vector that is compatible with the finite element formulation that approximates the distributed lateral force of gravity by equivalent lateral forces at the nodes, and moments at the nodes to approximate the interior loading of the beam elements.

This loading provides the opportunity to validate the equivalence of the state-space model of (27) with the second order model of (6) and (20). It further provides for a demonstration of the convergence of the time simulation provided by MATLAB®'s <lsim> command [25] . Finally, the validity of the final value theorem approximation using MATLAB®'s <ss2tf> command [25], as was done in section 4.2.4, may be demonstrated in a more complicated context.

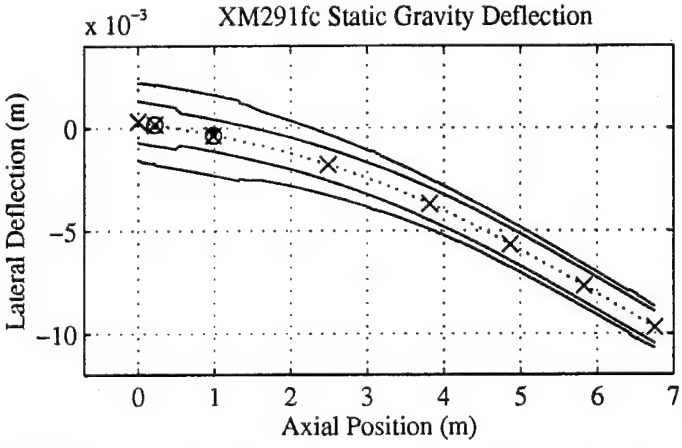


Figure 14 Gravity Deflection Computed in Sections 11 to 14 of <XM291fc.m>.

Figure (14) is the juxtaposition of the solution of the static deflection problem arrived at from three wholly different principles. No perceptible difference occurs in the plots, validating their similarity. Numerically, small —but not insignificant— differences do exist.

The first method used was a direct simulation of the response of the system to the instantaneous application of gravity loading for a relatively long period of time. A period sufficient for the motion to essentially cease (four seconds). This motion provided the opportunity to demonstrate the potential for animation of the response in section 12 of the m-file <XM291fc.m>.

The second method was to invoke the final value theorem by assembling a matrix of each input/output pair combination using two nested loops. At each pair, the MIMO state-space system was truncated to SISO system of the input/output pair. This system was converted to its transfer function equivalent using MATLAB®'s <ss2tf> command [25], and then evaluated with the Laplace "*s*" operator set to zero as was done in the previous section. This results in a matrix that transforms the generalized force vector, to the equilibrium generalized coordinate vector. This operation was executed in section 13 of the m-file <XM291fc.m>.

The third method used was to recognize that at equilibrium, the time derivatives of the generalized coordinates are zero. Thus, the second-order equations of motion, (6) and (20), can be simplified. The resulting solution for the deflection is the inverse of the stiffness matrix multiplied by the generalized force vector. This provides some insight into the relationship between the invertibility of the stiffness matrix, and the constraint of the beam that was elaborated on in section 3.4.1.

4.3 HYBRID 60MM TEST GUN

A 60mm test gun is currently being modified to support an electromagnetic accelerator at its muzzle end. [31] The accelerator unit is quite massive relative to the gun, and the dynamic effects of the electromagnetic reactive loading on such systems is a concern from the perspective of accuracy. [32] Finite element analysis of the barrel was conducted with, and without the muzzle accelerator, to reveal the mode-shapes of the system. These mode-shapes were compared with mode-shapes computed using the Uniform Segment Method (USM) discussed in section 1. [33]

4.3.1 BEAM GEOMETRY

The geometry of the 60mm test gun, with the extraneous muzzle accelerator masses is read in using the function file <geomf_hybrid.m> in a manner analogous to section 4.1.2. The plot of figure 15 reveals the simplified geometry of the test barrel, and the rather large extraneous mass that the muzzle accelerator imposes on the muzzle end of the barrel. Since the muzzle accelerator is both relatively short and stiff, it was not modeled as a beam element but rather as a series of rigid body masses applied at the muzzle end using the technique developed in section 2.3.2.

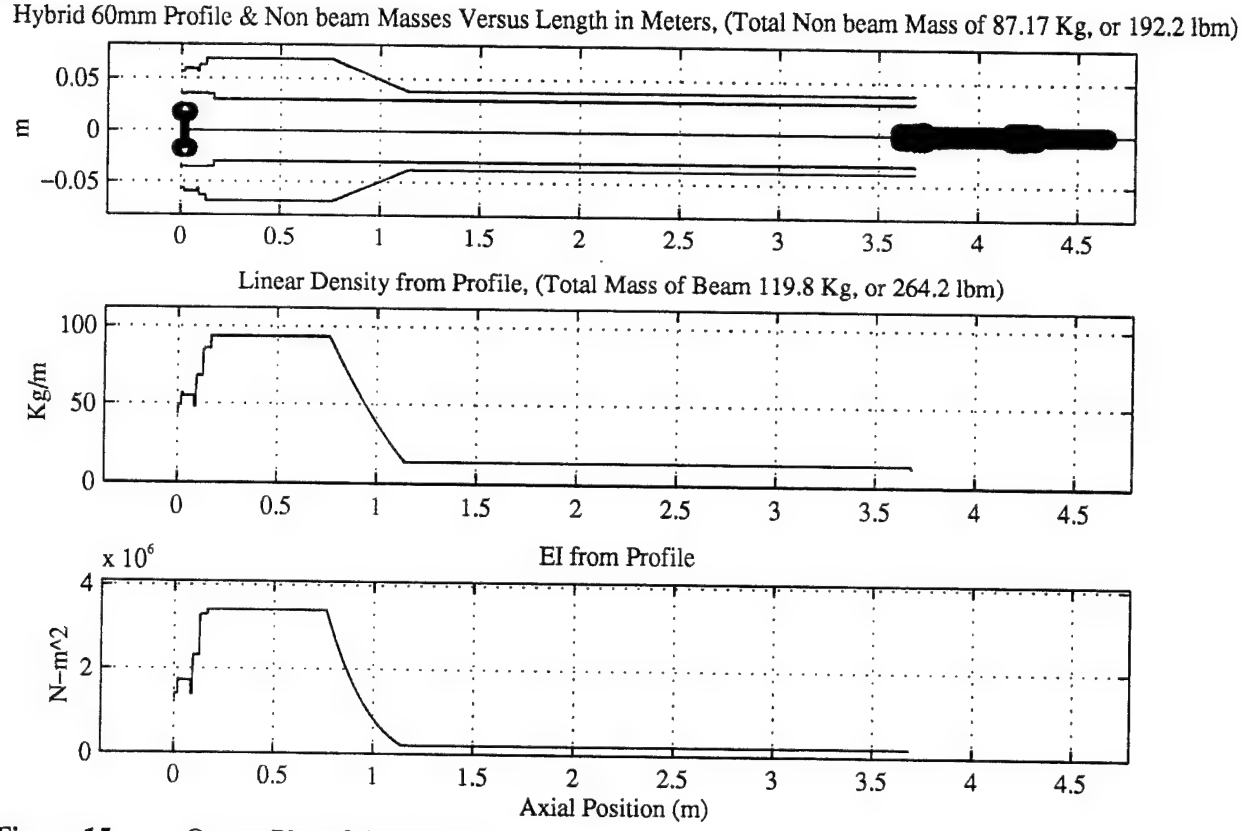


Figure 15 Output Plot of the M-file, <geomf_hybrid.m>.

4.3.2 UNDAMPED MODE-SHAPE COMPARISON

The mode-shapes of the free-free beam with out the accelerator and breech attached is shown in figure 16, juxtaposed against the mode-shapes computed using the USM. [33] For reasons of clarity and compatibility with the available USM eigenvectors, the mode-shapes are normalized to a maximum absolute value of one, and plotted against a normalized axis from zero to one. The similarity of eigen analysis of the two methods validates them.

It is worth noting that one of the key issues to using the USM is to reduce the model to a minimum number of elements that are *analytically* treated as uniform beam segments. Only three elements were employed by the USM to generate the eigenvectors of figure 16. Since the barrel essentially consists of a prismatic section beyond the taper at one meter, the USM and high density finite element eigenvectors very closely match beyond the taper. The discrepancies in the higher modes behind the taper may largely be attributed to the over simplification of the USM's first two elements.

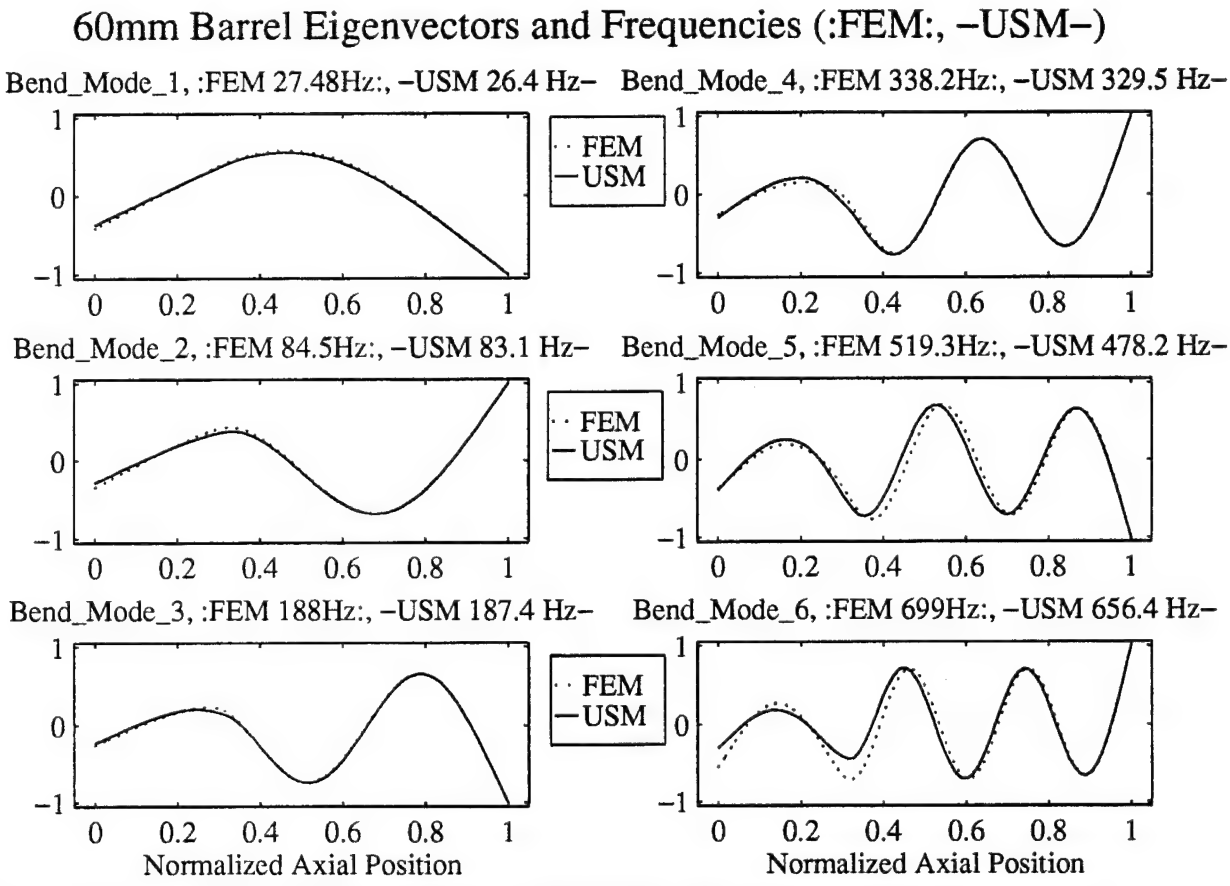


Figure 16 Hybrid Barrel Mode Shape Comparison Computed by <hybrid60.m>, Section 7.

4.3.3 GRAVITY DEFLECTION OF HYBRID GUN AS SUSPENDED

The deflections of the hybrid gun, as suspended in the firing range, is of interest and shown in figure 17. For the sake of clarity, the stiffness of the hangers was intentionally reduced to better depict the curvature of the barrel in relation to its suspension points.

This plot also reveals that many elements were used to generate the finite element model. (74 elements in total.) This is inferred from the x's plotted along the deformed centerline of the beam.

The deflection was computed using the third method of section 4.2.5; inversion of the stiffness matrix multiplied by the force vector.

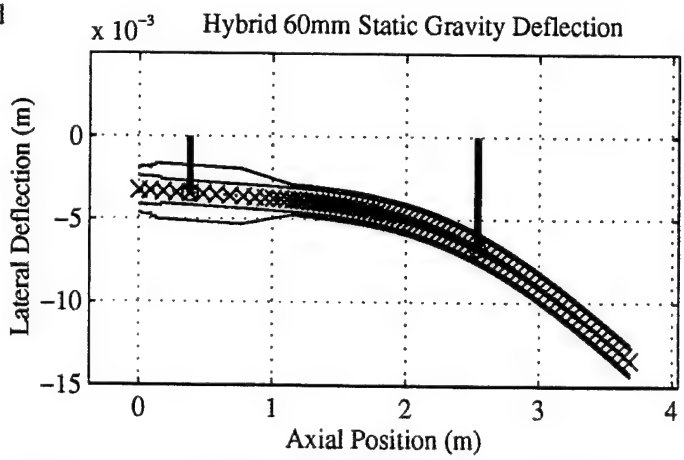


Figure 17 Gravity Droop Computed in Section 12 of <hybrid60.m>.

4.4 VALIDATION VIA COMPARISON WITH ANALYTIC CASES

4.4.1 NICHOLSON SOLUTION TO SPECIAL NON-UNIFORM BEAMS

Further validation of the modeling method developed in this report may be achieved by comparing the finite element approximations to the fundamental frequency of non-uniform beams of special cross-sections that have been analytically investigated with their analytic counterpart.

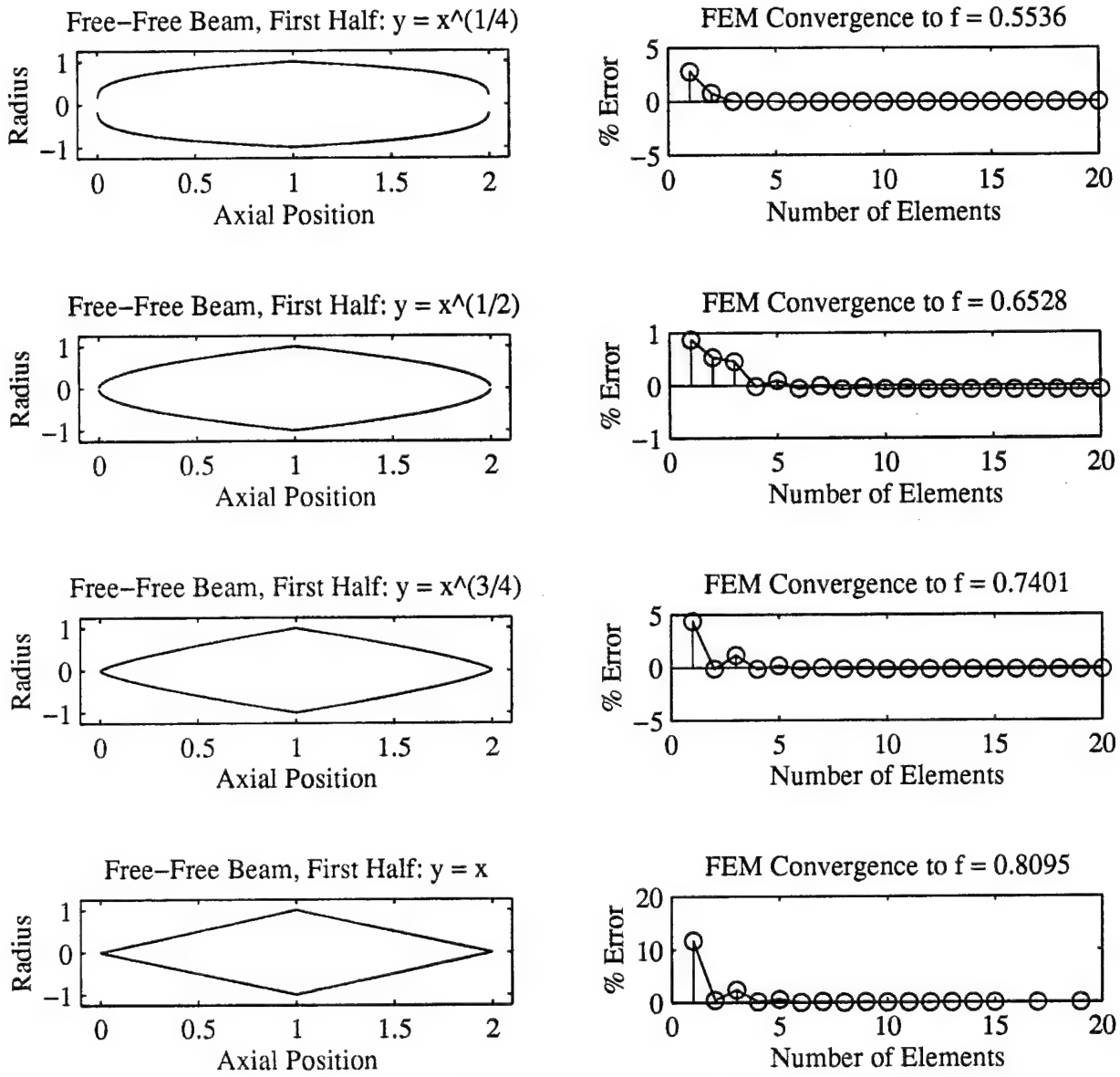


Figure 18 Comparison of Fundamental Frequencies to Analytic Solution.

Such analytic computations were computed by J. W. Nicholson (in the Proc. Roy. Soc. (London), 93, 1917, p. 506). [34] Figure 18 depicts the geometry of the beams analyzed, lists the analytic solution for unity values of density and elasticity, and plots the relative finite element error in the computation of fundamental frequencies versus the number of finite elements. The computations were executed in the m-file <Nicholson.m>. (Note: due to the zero taper of the beams at either end, numerical accuracy is decreased due to the extreme variation in the inertial and stiffness matrix diagonal elements. Use of the meshing file <fem_mesh.m> only exacerbated the problem, so it was not used.)

4.4.2 UNIFORM BEAM SOLUTION

The solution of the transverse bending beam equation using the Euler-Bernoulli approximation results in the following homogeneous differential equation of motion: [6, 15]

$$-\frac{\partial^2}{\partial x^2}\left[EI(x)\frac{\partial^2 y(x,t)}{\partial x^2}\right] = \rho(x)\frac{\partial^2 y(x,t)}{\partial t^2} \qquad 0 < x < L \tag{58}$$

This can be simplified using the separation of variables technique as in (29) and (43) by assuming a sinusoidal form of the solution for F(t): [6, 15]

$$\begin{aligned} y(x,t) &= Y(x)F(t) \\ \therefore \quad \frac{\partial^2 y(x,t)}{\partial t^2} &= -\omega^2 Y(x)F(t) \quad \textit{where } \omega \textit{ is the frequency of } F(t) \end{aligned} \tag{59}$$

Finally, in the case of uniform beams, EI(x) and ρ(x) are constants:

$$\begin{aligned} \frac{\partial^2}{\partial x^2}\left[EI(x)\frac{\partial^2 Y(x)}{\partial x^2}\right] &= \omega^2 \rho(x) Y(x) \\ \frac{\partial^4 Y(x)}{\partial x^4} - \frac{\omega^2}{c^2}Y(x) &= 0 \\ \textit{where: } c^2 &= \frac{EI}{\rho} \end{aligned} \tag{60}$$

The spatial solution admits functions of the form, $Y(x) = e^{kx}$. This leads to the general solution: [15]

$$\begin{aligned} Y(x) &= C_1\big(\cos(kx) + \cosh(kx)\big) + C_2\big(\cos(kx) - \cosh(kx)\big) \\ &\quad + C_3\big(\sin(kx) + \sinh(kx)\big) + C_4\big(\sin(kx) - \sinh(kx)\big) \\ \textit{where } k &= \sqrt{\frac{\omega}{c}} \end{aligned} \tag{61}$$

Where k, C_1, C_2, C_3, and C_4 are undetermined constants. The boundary conditions for a free-free beam imply that both the moment and shear forces vanish at the ends of the beam (x = 0 and x = L). This further implies that the second and third spatial derivatives of Y(x) are zero at the ends of the beam respectively. Using the assumed form for the spatial solution of (61), the following four boundary conditions may be applied:

$$
\begin{aligned}
\frac{\partial^2 Y(x)}{\partial x^2}\Big|_{x=0} &= -2C_2k^2 = 0 \qquad \frac{\partial^3 Y(x)}{\partial x^3}\Big|_{x=0} = -2C_4k^3 = 0 \\
\frac{\partial^2 Y(x)}{\partial x^2}\Big|_{x=L} &= C_1\big(-\cos(kL)+\cosh(kL)\big)k^2 + C_2\big(-\cos(kL)-\cosh(kL)\big)k^2 \\
&\quad + C_3\big(-\sin(kL)+\sinh(kL)\big)k^2 + C_4\big(-\sin(kL)-\sinh(kL)\big)k^2 = 0 \\
\frac{\partial^3 Y(x)}{\partial x^3}\Big|_{x=L} &= C_1\big(\sin(kL)+\sinh(kL)\big)k^3 + C_2\big(\sin(kL)-\sinh(kL)\big)k^3 \\
&\quad + C_3\big(-\cos(kL)+\cosh(kL)\big)k^3 + C_4\big(-\cos(kL)-\cosh(kL)\big)k^3 = 0
\end{aligned}
\tag{62}
$$

From (62) it is clear that C_2 and C_4 are zero. Solving for C_1 and C_3:

$$
\begin{bmatrix} \big(-\cos(kL)+\cosh(kL)\big) & \big(-\sin(kL)+\sinh(kL)\big) \\ \big(\sin(kL)+\sinh(kL)\big) & \big(-\cos(kL)+\cosh(kL)\big) \end{bmatrix} \times \begin{bmatrix} C_1 \\ C_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{63}
$$

(63) is an eigenvalue equation. The valid set of eigenvalues satisfies the equation formed by setting the determinate to zero. (The trigonometry identities of $\sin^2 + \cos^2 = 1$ and $\sinh^2 + \cosh^2 = 1$ are required for the simplified form below that was evaluated using the <simple> and <determ> commands. [13]):

$$
\begin{vmatrix} \big(-\cos(kL)+\cosh(kL)\big) & \big(-\sin(kL)+\sinh(kL)\big) \\ \big(\sin(kL)+\sinh(kL)\big) & \big(-\cos(kL)+\cosh(kL)\big) \end{vmatrix} = 2\big(1-\cos(kL)\cosh(kL)\big) = 0 \tag{64}
$$

An infinity of solutions exist to (64), these correspond to the modal frequencies. In the context of finite element analysis, this infinity of solutions may be thought of as the limit of breaking the beam up into an infinite number of infinitesimal segments as was expressed by Simpson's hypothesis in section 2.1. [8] As in the spatially discretized case, the higher modes are physically subject to disproportionately higher damping ratios, and are difficult to excite via mechanical systems. For these reasons, at some —*application specific*— cut-off frequency, the higher modes may be determined to have a negligible affect on the dynamic response of the system in the operating range of interest. (In the case of gun systems, it has been argued that this cut-off would occur at the sixth flexible mode. [3])

The mode-shape of any modal frequency may be determined by solving (63) using the frequency solution of (64). This could be done assuming a value of unity for C_3, and solving for C_1 using the second row of (63). This completes all four unknown constants in (61). For the non-dimensional case, E, I, ρ, and L are all unity, with $0 < x \leq 1$. This results in the following mode shape solution:

$$
Y(x) = -\left(\frac{-\cos(\sqrt{\omega})+\cosh(\sqrt{\omega})}{\sin(\sqrt{\omega})+\sinh(\sqrt{\omega})}\right)\big(\cos(\sqrt{\omega}x)+\cosh(\sqrt{\omega}x)\big) + \big(\sin(\sqrt{\omega}x)+\sinh(\sqrt{\omega}x)\big) \tag{65}
$$

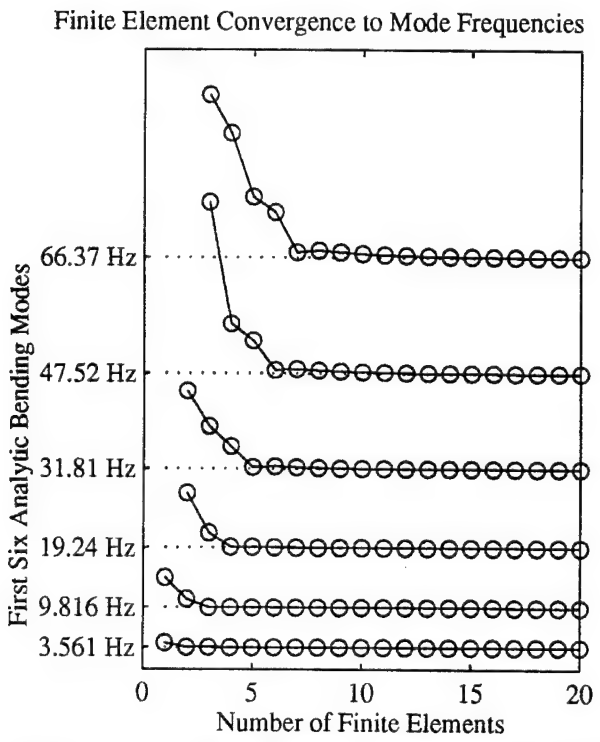


Figure 19 Finite Element Convergence to Analytic Frequencies.

The results of equations (64) and (65) were implemented in <uniform.m>. The formation of a function file <ubeameig.m> was required to define the eigenvalue function of (64) to facilitate the use of MATLAB®'s <fzero> [12] root finding function. The eigenvalues were then used to define the mode-shapes of (65). The analytic results were then compared with finite element modeling of a normalized beam. The convergence depicted by figure 19 demonstrates that the lower modes converge faster than the higher modes.

Figure 20 is a plot of the estimated mode-shapes versus the analytic solution. Two finite element approximations are shown. The first employed a mere three elements in its approximation of the beam. (Note from figure 19, that the three element approximation is the lowest number of elements that will approximate the first six bending modes.)

The second approximation used twenty finite elements. At this resolution, the dotted line that represents the high resolution finite element approximation is coincident and indistinguishable from the solid line that represents the analytic solution for all six modes displayed.

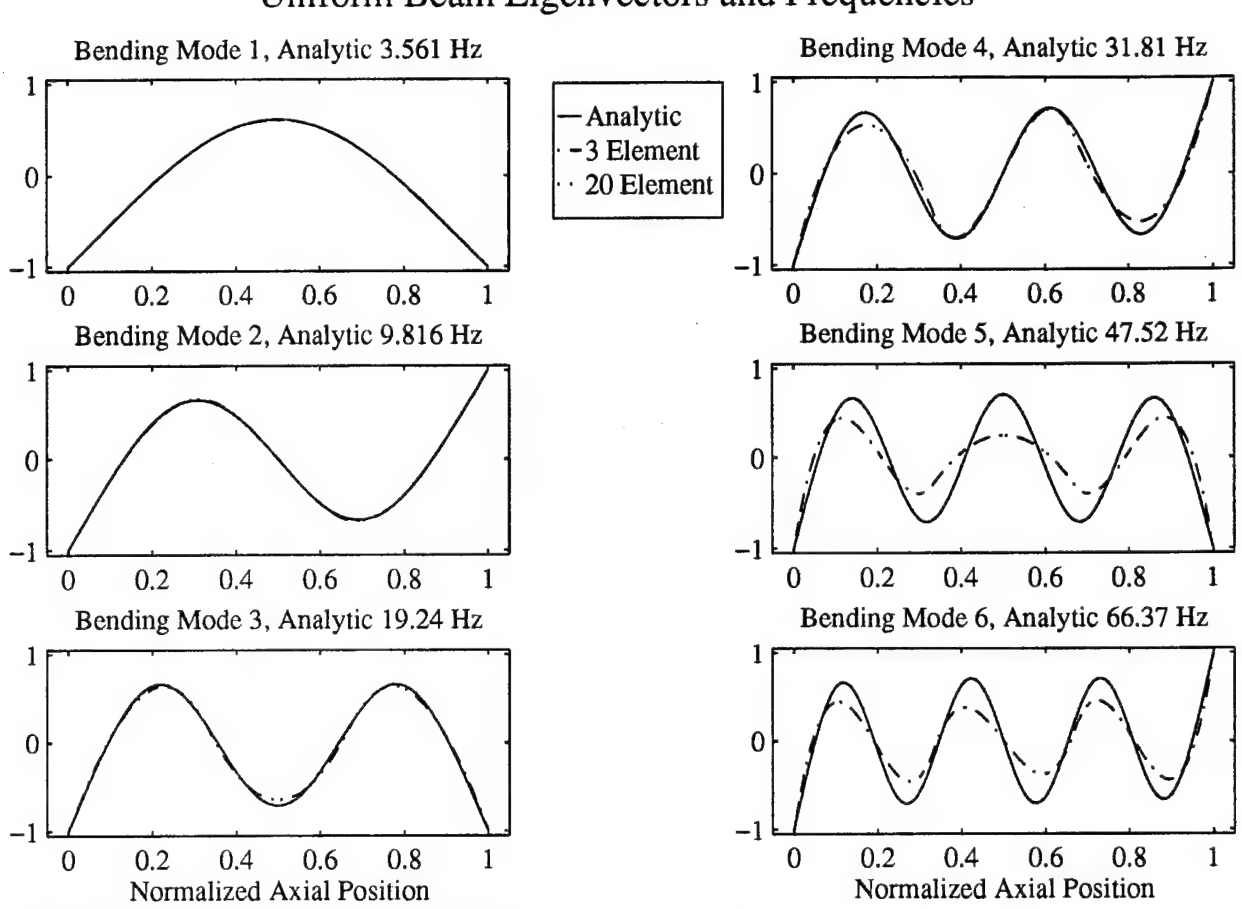


Figure 20 Juxtaposition of FEM Approximations and the Analytic Mode-Shapes.

5 CONCLUSIONS

This report has documented the development of finite element modeling of non-uniform beam dynamics within the MATLAB® software environment. Several objectives have been achieved. First, the report may serve as a detailed users guide for others who may wish to use the software developed within this report. Second, the principles behind the mathematical modeling have largely been laid down in detail, so that the reader may understand why the modeling approach works. A large variety of references are cited to allow the interested reader to dig deeper in this regard. Third, several demonstrations of the utility of working within MATLAB® are provided to illustrate some of the powerful analysis tools that have been enabled. Fourth, several case-studies have been conducted to demonstrate how the software is used, and to document some of the results of the analysis that justified the development of this code. Finally, the results of the modeling developed has been compared with the results of analytic methods, validating the accuracy of the approach developed in this effort.

6 APPENDIX

<Nicholson.m>

```
% Nicholson.m -> A working script M-file to execute analyses of non-
% uniform beams as investigated by J. W. Nicholson
% (in the Proc. Roy. Soc. (London), 93, 1917, p. 506) and communicated by
% Timoshenko, Vibration Problems in Engineering, p 393 of 3rd, and p 473
% of 5th edition.
% o  Created 09 January 1996 by Eric Kathe.
%   Benet Labs, Watervliet Arsenal, NY  12189-4050 <ekathe@pica.army.mil>
% ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

% ______________________________________________________________________________
% Section (1)
% Compute approximations of the beam geometry for the five cases listed
% in Timoshenko.  y = a*x^m where m = 0, 1/4, ½, 3/4, 1.  Also
% compute analytic fundamental frequencies.
% DEFINE: plot_on, print_on, tlabel, spatial, lden, lEI, lnbden, Mextl,
%         Mextr, gm.
% ..........................................................................
% Set plot and print flags to 1 to enable, zero to disable.
plot_on = 1;
print_on = 0;
%
if print_on == 1
  plot_on = 1;  % Clearly, to print, the plot flag must be enabled.
end
%
% Define the sampled radius vectors as the function of a, x, and m.
%
l = 1;
a = 1;
m = [1/4 ½ 3/4 1]';
nm = length(m);
x = (1:1001)'/1001;
r = zeros(2001,nm);
for I = 1:nm
  y = a*x.^m(I);
  r(1:1001,i) = y;
  r(1002:2001,i) = flipud(y(2:1001));% Leaves one max value @ 1.
end
%
% Set material properties to unity.
%
rho = 1;
E = 1;
%
% Read in alpha values for the five cases and compute
% the analytic frequencies.
%
alpha = [6.957 8.203 9.300 10.173]';
f = alpha*max(y)/(4*pi*l^2)*sqrt(E/rho);
%
% ______________________________________________________________________________
% Section (2)
%
if plot_on == 1
  figure(1)
  clg
  set(gcf,'PaperOrientation','portrait'); % This series of commands configures
  set(gcf,'PaperUnits','inches');        % the plot window to effectively
  set(gcf,'PaperPosition',[1 1 6.5 6.5]); % be incorporated into a report.
  set(gcf,'Units','inches','position',get(gcf,'PaperPosition'));
  set(gcf,'DefaultAxesFontSize',10);
  %
  for I = 1:nm
    spatial = [x; 1+x(1:1000)];
    lden = rho*pi*(r(:,I).^2);
    lEI = E*(r(:,I).^4)*pi/4;
    lnbden = zeros(size(lden));
    gm = r(:,I);
    subplot(nm,2,(1+2*(I-1)))
    plot(spatial,gm,'k',spatial,-gm,'k')
    [num,den] = rat(m(I));
    if den > 1
      mstr = ['^(' int2str(num) '/' int2str(den) ')'];
    else
      if num == 1
        mstr = [];
      else
        mstr = ['^' int2str(num)];
      end
    end
    title(['Free-Free Beam, First Half: ' ...
      'y = x' mstr ])
    xlabel('Axial Position')
    ylabel('Radius')
    axis([-0.1 2.1 -1.25 1.25])
    shrink = 0.5;
    pos = get(gca,'position'); % This is in normalized coordinates
    pos(2)=pos(2) + pos(4)*(shrink/2);% Raise the subplot by the saved height.
    pos(4)=pos(4)*(1-shrink);       % Shrink the height by a shrink factor.
    set(gca,'position',pos);
    %
    for j = 1:20
      ncord = (1:j)'/j;
      ncord = [1; round(2001*ncord)];
      [Mfem,Kfem] = fem_form(spatial,lden,lEI,lnbden,ncord);
      [phi,fvn,rlab] = eigen_2o(Mfem,Kfem);% Compute, undamped eigen frequencies.
      ffem(i,j) = abs(fvn(3));% Because of the poorly conditioned ends, small
                % imaginary content is common.
    end
    perror = (ffem(I,:) - f(I))/f(I)*100;% Compute percent error.
    subplot(nm,2,(2*I))
    plot(perror)
    hold on
    stem(perror)
    hold off
    title(['FEM Convergence to f = ' num2str(f(I))])
    xlabel('Number of Elements')
    ylabel('% Error')
    pos = get(gca,'position'); % This is in normalized coordinates
    pos(2)=pos(2) + pos(4)*(shrink/2);% Raise the subplot by the saved height.
    pos(4)=pos(4)*(1-shrink);       % Shrink the height by a shrink factor.
    set(gca,'position',pos);
  end
  subtitle('Comparison of Fundamental Mode to Analytic Solution')
end
%
if print_on == 1
  print -deps fig18.ps
end
% ______________________________________________________________________________
% Completed:
% ..........................................................................
```

<XM291fc.m>

```
% XM291fc.m -> A working script M-file to execute analyses of a fully
% constrained XM291 using the finite element formulation and dynamics
% analysis functions defined within the subdirectory, beam_fem/.
% o  Created 24 October 1995 - 08 January 1996 by Eric Kathe.
%   Benet Labs, Watervliet Arsenal, NY  12189-4050 <ekathe@pica.army.mil>
% ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

% ______________________________________________________________________________
% Section (1)
% Set the plot and print flags on or off, run the geometry
% m-file, and create a plot label.
% DEFINE: plot_on, print_on, tlabel, spatial, lden, lEI, lnbden, Mextl,
%         Mextr, gm.
% ..........................................................................
%
% Set plot and print flags to 1 to enable, zero to disable.
plot_on = 1;
print_on = 0;
%
if print_on == 1
  plot_on = 1;  % Clearly, to print, the plot flag must be enabled.
end
%
tlabel = 'XM291fc';% tlabel -> Enables plotting.
%
% Load in geometry data as in XM291rb.m:
[spatial, lden, lEI, lnbden, Mextl, Mextr, gm] = geomf_XM291;
%
% ______________________________________________________________________________
% Section (2)
% Define barrel constraint locations, number of elements, and create
% element mesh vector.
% DEFINE: snlv, ncord.
% USE: spatial, lden, lEI, lnbden, snlv, tlabel.
% ..........................................................................
%
elmechloc = 0.220;% (m) Mid-span of breech ring threads on barrel.
trunnionloc = 38.9*(0.0254);% in(m/in) Yoke position as in geomf_XM291
%                 line 462.
```

```
snlv = [elmechloc; trunnionloc]% (m) Imposed node location vector.
%
nel = 7;% A relatively small number of FEM elements to keep the
%       problem quick & manageable for initial controls work.
%
%
% Generate meshing vector as in XM291rb.m:
[ncord] = fem_mesh(spatial,lden,lEI,lnbden,snlv,nel);
%
elmechloc = []; trunnionloc = []; nel = [];
%
% ____________________________________________________________
% Section (3)
% Generate the mass and stiffness matrices via FEM.
% DEFINE: Mfem, Kfem.
% USE: spatial, lden, lEI, lnbden, ncord, tlabel
% ..........................................................
%
[Mfem,Kfem] = fem_form(spatial,lden,lEI,lnbden,ncord);
% ____________________________________________________________
% Section (4)
% Generate the constraint matrix for fully constrained XM291.
% (This will include the elevation mechanism and the trunnion, with
% the elevation mechanism stiffness estimated at 2/3's the value of the
% trunnion constraints.)
% Also predefine the alpha and beta for Rayleigh
% damping. (Shames & Dyn, pp. 646.)
% DEFINE: alpha, beta, constraintm.
% ..........................................................
%
alpha = 0.000; % Mass proportional damping factor.
beta = 0.001;  % Stiffness proportional damping factor.
%
kcnst = [(2/3);1]*750000*(4.4482)/(0.0254);% (lbf/in)(N/lbf)/(m/in)
% Estimated lateral elasticity of the Elevation and Trunnion constraints.
%
cdcnst = beta*kcnst; % Stiffness proportional damping of constraints.
%
[y, k1el] = min(abs(snlv(1) - spatial(ncord)));% Identification of
%                                   constraint node number.
[y, k1tr] = min(abs(snlv(2) - spatial(ncord)));% Identification of
%                                   constraint node number.
gcindk = 2*[k1el;k1tr] - 1;% Identification of constraint generalized
%                   coordinate number.
%
constraintm = [gcindk kcnst cdcnst];
%
kcnst = []; cdcnst = []; y = []; k1tr = []; gcindk = [];
% ____________________________________________________________
% Section (5)
% Lump in the external masses and constraints into the Mass and Stiffness
% matrices.  Also generate the Rayleigh damping matrix.
% DEFINE: M, K, Cd, n2.
% USE: Mfem, Kfem, Mextl, Mextr, alpha, beta, constraintm.
% ..........................................................
%
[M,K,Cd] = fem_lump(Mfem,Kfem,Mextl,Mextr,alpha,beta,constraintm);
%   M, K, Cd -> Mass, stiffness, and Rayleigh damping matrices of
%       generalized coordinates that include the constraint and
%       externally coupled dynamics.
%
n2 = size(M,1);% The number of generalized coordinates.
% ____________________________________________________________
% Section (6)
% Generate the second order, damped eigen modes.
% DEFINE: phi, fv, fvn, rlab.
% USE: author supplied data.
% ..........................................................
%
[phi,fvn,rlab] = eigen_2o(M,K);% Compute, undamped eigen frequencies.
[phi,fv,rlab] = eigen_2o(M,K,Cd);% Compute, normalize, sort, and identify
%                           the modes of vibration.
if plot_on == 1
 figure(10)
 clg
 set(gcf,'PaperOrientation','portrait');  % This series of commands configures
 set(gcf,'PaperUnits','inches');          % the plot window to effectively
 set(gcf,'PaperPosition',[1 1 3.25 5.5]);  % be incorporated into a report.
 set(gcf,'Units','inches','position',get(gcf,'PaperPosition'));
 set(gcf,'DefaultAxesFontSize',9);
 for n = 1:4
    subplot(4,1,n)
    beam_plot(ncord,spatial,phi(:,n),gm/5,constraintm(:,1));
    mlabel = rlab(n,find(abs(rlab(n,:)) ~= 32));
    if strcmp(rlab(n,1),'B') % Check to see if bending mode.
     title([mlabel ', ' num2str(fv(n)) 'Hz'])
    else % Else -> Rigid body with near-zero frequency not displayed.
     title([mlabel])
    end
    axis([-0.2 7 -.1 .1])
    grid
    if n == 4
     xlabel('Axial Position (m)')
    end
    pos = get(gca,'position'); % This is in normalized coordinates
    pos(4)=pos(4)*.85;        % Shrink the height by a factor of .85
    pos(2)=pos(2) + pos(4)*.15;% Raise the subplot by the saved height.
    set(gca,'position',pos);
 end
 subtitle([tlabel ' Damped Mode-Shapes'])
end
%
if print_on == 1
 print -deps fig10.ps; % Print the file as an encapsulated Post-Script file.
end
%
n = []; k = []; I = []; mlabel = [];
%
% ____________________________________________________________
% Section (7)
% Convert to first-order state-space and truncate to muzzle, trunnion SISO.
% DEFINE: A, B, C, D, nq, states, outputs, inputs, ae, be, ce, de.
% USE: M, K, Cd.
% ..........................................................
%
[A,B,C,D] = fem2ss(M,K,Cd);% This form assumes F1 through Mn inputs
%                   and all generalized coordinates as outputs.
[ae,be,ce,de] = ssselect(A,B,C,D,5,n2);% F3 -> gc(5).
%
% ____________________________________________________________
% Section (8)
% Bode Plot.
% USE: ae, be, ce, de.
% ..........................................................
%
[mag,phase,w] = bode(ae,be,ce,de,1,logspace(log10(5*2*pi),log10(500*2*pi),500));
if plot_on == 1
 figure(11)
 clg
 set(gcf,'PaperOrientation','portrait');  % This series of commands configures
 set(gcf,'PaperUnits','inches');          % the plot window to effectively
 set(gcf,'PaperPosition',[1 1 6.3 3]);    % be incorporated into a report.
 set(gcf,'Units','inches','position',get(gcf,'PaperPosition'));
 set(gcf,'DefaultAxesFontSize',9);
 whitebg(gcf,'k')
 subplot(211), semilogx(w/(2*pi),20*log10(mag))
 title([tlabel ' Trunnion Disturbance to Muzzle Pointing Angle Bode Plot'])
 ylabel('Gain dB')
 av = [ [min(w) max(w)]/(2*pi) min(20*log10(mag)) 0.95*max(20*log10(mag))];
 axis(av)
 fabsissa = sort([ fv(1:4); 500]);
 flabel = [];
 for I = 1:length(fabsissa)
  if I == 4
   flabel = str2mat(flabel,['     ' freq2str(fabsissa(I),1)]);
  else
   flabel = str2mat(flabel,freq2str(fabsissa(I),1));
  end
 end
 flabel(1,:) = [];
 set(gca,'XTick',fabsissa)
 set(gca,'XTickLabels',flabel)
 set(gca,'YTick',[min(20*log10(mag)), median(20*log10(mag)), max(20*log10(mag))])
 grid
 subplot(212), semilogx(w/(2*pi),phase)
 xlabel('Frequency (Hz)'), ylabel('Phase deg')
 av = [ [min(w) max(w)]/(2*pi) min(phase) (max(phase)+180)];
 axis(av)
 set(gca,'YTick',[180:-180:-600])
 grid
end
%
if print_on == 1
 set(gcf,'InvertHardCopy','on')
 print -deps fig11.ps; % Print the file as an encapsulated Post-Script file.
end
%
% ____________________________________________________________
% Section (9)
% Impulse Response.
% USE: ae, be, ce, de.
% ..........................................................
```

```
%
[y,x,t] = impulse(ae,be,ce,de,1,[0:0.000005:0.4]);
%
if plot_on == 1
 figure(12)
 clg
 set(gcf,'PaperOrientation','portrait');  % This series of commands configures
 set(gcf,'PaperUnits','inches');          % the plot window to effectively
 set(gcf,'PaperPosition',[1 1 3.8 3.5]);  % be incorporated into a report.
 set(gcf,'Units','inches','position',get(gcf,'PaperPosition'));
 set(gcf,'DefaultAxesFontSize',9);
 plot(t,y,'k',[min(t) max(t)],[0 0], 'k:')
 av = axis;
 av(3) = 2.2*av(3);
 axis(av)
 pos = get(gca,'position');        % This is in normalized coordinates
 pos(2)= pos(2) + (1-0.85)*pos(4)/2; % Raise the subplot but keep it centered.
 pos(4)= 0.85*pos(4);              % Shrink the height by a factor of .85
 set(gca,'position',pos);
 title('          SISO Fully Constrained Impulse Response')
 ylabel('Muzzle Deflection (rad)')
 xlabel('Time After Newton-Second Impulse at Trunnion (sec)')
 newpos = [(pos(1)+pos(3)/3) (pos(2)+pos(4)/7) (pos(3)/2) (pos(4)/3)];
 axes('position',newpos)
 ttrunk = t(1:1500);
 plot(ttrunk,y(1:length(ttrunk)),'k',[0 0.010],[0 0],'k:')
 av = [0 0.01 1.1*[min([0;y(1:length(ttrunk))]) max(y(1:length(ttrunk)))]];
 axis(av)
 title('              Initial Response Close-Up')
 ylabel('(rad)')
 set(gca,'XTick',[0 0.005 0.010])
 set(gca,'XTickLabels',str2mat('0.000','0.005','0.010'))
 xlabel('(sec)')
end
%
if print_on == 1
 print -deps fig12.ps; % Print the file as an encapsulated Post-Script file.
end
%
% _____________________________________________________________
% Section (10)
% Unit Step Response.
% USE: ae, be, ce, de.
% ..........................................................
%
[y,x,t] = step(ae,be,ce,de,1,[0:0.00005:0.5]); % Compute step response.
[num,den] = ss2tf(ae,be,ce,de);  % Compute Transfer Function Polynomials.
fvy = num(length(num))/den(length(den)); % Final Value Theorem of TF.
%                                 Effectively sets a to zero.
if plot_on == 1
 figure(13)
 clg
 set(gcf,'PaperOrientation','portrait');  % This series of commands configures
 set(gcf,'PaperUnits','inches');          % the plot window to effectively
 set(gcf,'PaperPosition',[1 1 3.8 3.5]);  % be incorporated into a report.
 set(gcf,'Units','inches','position',get(gcf,'PaperPosition'));
 set(gcf,'DefaultAxesFontSize',9);
 plot(t,y,'k',[min(t) max(t)],[0 0], 'k:',[min(t) max(t)],fvy*[1 1], 'k-.')
 set(gca,'YTick',[max(y) 0 fvy min(y)])
 av = axis;
 av(3) = 2.2*av(3);
 axis(av)
 pos = get(gca,'position');        % This is in normalized coordinates
 pos(2)=pos(2)+(1-0.95)*pos(4)/2; % Raise the subplot & keep it centered.
 pos(4)=0.95*pos(4);              % Shrink the height by a factor of .95
 pos(1)=pos(1)+(1-0.85)*pos(4)/2; % Shift to the right.
 set(gca,'position',pos);
 title('          SISO Fully Constrained Step Response')
 ylabel('Muzzle Deflection (rad)')
 xlabel('Time After Newton Applied at Trunnion (sec)')
 newpos = [(pos(1)+pos(3)/3) (pos(2)+pos(4)/8) (pos(3)/3)*1.5 (pos(4)/4)];
 axes('position',newpos)
 ttrunk = t(1:200);
 plot(ttrunk,y(1:length(ttrunk)),'k',[0 0.015],[0 0],'k:')
 av = [0 0.01 1.1*[min([0;y(1:length(ttrunk))]) max(y(1:length(ttrunk)))]];
 axis(av)
 title('              Initial Response Close-Up')
 ylabel('(rad)')
 set(gca,'XTick',[0 0.005 0.010 0.015])
 set(gca,'XTickLabels',str2mat('0.000','0.005','0.010','0.015'))
 xlabel('(sec)')
end
%
if print_on == 1
 print -deps fig13.ps; % Print the file as an encapsulated Post-Script file.
end
% _____________________________________________________________
% Section (11)
% Force due to gravity.
% USE: A, B, C, D.
% ..........................................................
g = -9.8067; % Earth Surface Gravity (m/(s^2)).
weight = g*(lden + lnbden)*diff(spatial(1:2));
%
% [F] = fem_force(spatial,weight,zeros(size(lden)),ncord,tlabel);% Prints F.
[F] = fem_force(spatial,weight,zeros(size(lden)),ncord);
%
% Now incorporate the two 2x2 rigid body mass matrices after checking to
% avoid divide by zero:
%
if abs(Mextl(1,1)) > 0
 F(1) = F(1) + g*Mextl(1,1); % g*Kg.
 F(2) = F(2) + g*(Mextl(1,2)^2/Mextl(1,1)); % g*( (m*Kg)^2/Kg).
end
%
if abs(Mextr(1,1)) > 0
 F(n2-1) = F(n2-1) + g*Mextr(1,1);
 F(n2) = F(n2) + g*(Mextr(1,2)^2/Mextr(1,1)); % g*( (m*Kg)^2/Kg).
end
% _____________________________________________________________
% Section (12)
% Time simulation of deflection due to gravity.
% USE: A, B, C, D, F, M, K, n2.
% ..........................................................
%
t = (0:0.005:4)';   % Generate a time vector for the simulation.
u = ones(size(t))*F'; % Apply the same force vector at each time slice.
[y,x] = lsim(A,B,C,D,u,t); % Compute response.
tim = length(t);
xgs = x(tim,:);% X due to g at near infinity from simulation.
%
animate_on = 0;% Flag to execute animation of step response of gravity.
%
if animate_on == 1
 xlat = x(:,[1:2:n2]); % Identify the greatest lateral deflection.
 maxlat = max(max(xlat));
 minlat = min(min(xlat));
 scale = max(abs([maxlat minlat]))/(5*max(max(gm)));% Scale the beam plot.
 av = [ (min(spatial)- 0.1*max(spatial)) 1.1*max(spatial) ...
      (minlat - scale*min(min(gm))) (maxlat + scale*max(max(gm)))];
 %
 figure(14)
 clg
 set(gcf,'PaperOrientation','portrait');  % This series of commands configures
 set(gcf,'PaperUnits','inches');          % the plot window to effectively
 set(gcf,'PaperPosition',[1.75 3.75 5 3.5]);  % be incorporated into a report.
 set(gcf,'Units','inches','position',get(gcf,'PaperPosition'));
 set(gcf,'DefaultAxesFontSize',12);
 for I = 1:1:100
  clg
  beam_plot(ncord,spatial,x(i,1:size(M,1)),gm*scale,constraintm(:,1));
  title(['          Step Gravity Response Animation'])
  xlabel('Axial Position (m)')
  ylabel('Lateral Deflection (m)')
  hold on
  plot([min(spatial) max(spatial)],[0 0],'k:')
  hold off
  axis(av)
  pos = get(gca,'position');        % This is in normalized coordinates
  newpos = [(pos(1)+pos(3)/7) (pos(2)+pos(4)/7) (pos(3)/2) (pos(4)/3)];
  axes('position',newpos)
  plot(t(1:tim),y(1:tim,n2),'k')
  title(['          t = ' num2str(t(I)+10^(-7),3,1) 'sec'])
  % num2fstr.m is a modified form of num2str.m that imposes the format
  % notation of %5.3f to keep the time from bouncing.  (It's easy to do.)
  ylabel('  Muzzle Deflection (rad)')
  xlabel('Time (s)')
  axis([min(t) max(t) round(min(y(1:tim,n2)*(10^4)))/10^4 ...
          round(max(y(1:tim,n2))*(10^4))/10^4])
  hold on
  plot(t(I),y(i,n2),'bo')
  hold off
  % Create box around entire plot region:
  exterior = [0 0 1 1];
  axes('position', exterior)
  set(gca,'Box','on')
  set(gca,'XTick',[])
  set(gca,'YTick',[])
  %
  % Uncomment to save images as sequential gif files to be read into
  % standard movie maker package.  (This is the best way to go.)
  %
```

```
384   % eval(['print -dgif animate' int2str(I) '.gif'])
385   pause(.5)
386  end
387 end
388 % ____________________________________________________________
389 % Section (13)
390 % Final Value Theorem Prediction of static deflection due to gravity.
391 % USE: ae, be, ce, de.
392 % ..........................................................
393 %
394 FVTm = zeros(size(M));% Initialize matrix to zeros.
395 %
396 % Loop through all of the indices to evaluate the final value theorem
397 % of the transfer function.
398 %
399 for I = 1:size(M,1)
400  for j = 1:size(M,1)
401   [a,b,c,d] = ssselect(A,B,C,D,i,j);
402   [num,den] = ss2tf(a,b,c,d);  % Compute Transfer Function Polynomials.
403   fvy = num(length(num))/den(length(den)); % Final Value Theorem of TF.
404   %                          Effectively sets s to zero.
405   FVTm(j,i) = fvy;
406  end
407 end
408 %
409 xfvt = FVTm*F;
410 % ____________________________________________________________
411 % Section (14)
412 % Second order inverted stiffness approximation to static gravity loading.
413 % USE: K and F.
414 % ..........................................................
415 xinvk = KVF;% Ie, since x ddot -> zeros, K*x = F -> x = inv(K)*F.
416 % ____________________________________________________________
417 % Section (15)
418 % Juxtapose three methods to determine static gravity deflection on one
419 % plot.
420 % USE: xgs, xfvt, xinvk, gm, spatial, M.
421 % ..........................................................
422 sm = 2*floor(size(M,1)/2);
423 xtemp = xfvt(1:2:sm);
424 scale = max(abs(xtemp))/(5*max(max(gm)));
425 av = [ (min(spatial)- 0.1*max(spatial)) 1.1*max(spatial) ...
426      min(xtemp) max(xtemp)];
427 %
428 if plot_on == 1
429  figure(14)
430  clg
431  set(gcf,'PaperOrientation','portrait');  % This series of commands configures
432  set(gcf,'PaperUnits','inches');          % the plot window to effectively
433  set(gcf,'PaperPosition',[1.75 3.75 4 2.5]);  % be incorporated into a report.
434  set(gcf,'Units','inches','position',get(gcf,'PaperPosition'));
435  set(gcf,'DefaultAxesFontSize',10);
436  beam_plot(ncord,spatial,xfvt,gm*scale,constraintm(:,1));
437  hold on
438  beam_plot(ncord,spatial,xinvk,gm*scale,constraintm(:,1));
439  beam_plot(ncord,spatial,x(length(t),1:size(M,1)),gm*scale,constraintm(:,1));
440  hold off
441  av = axis;
442  av(1) = -av(2)/10;
443  axis(av)
444  grid
445  title(['      ' tlabel ' Static Gravity Deflection'])
446  xlabel('Axial Position (m)')
447  ylabel('Lateral Deflection (m)')
448  pos = get(gca,'position');          % This is in normalized coordinates
449  pos(2)=pos(2)+(1-0.85)*pos(4)/2; % Raise the subplot & keep it centered.
450  pos(4)=0.85*pos(4);                % Shrink the height by a factor of .85
451  pos(1)=pos(1)+(1-0.85)*pos(4)/2; % Shift to the right.
452  set(gca,'position',pos);
453 end
454 %
455 if print_on == 1
456  print -deps fig14.ps; % Print the file as an encapsulated Post-Script file.
457 end
458 % ____________________________________________________________
459 % Completed:
460 % ..........................................................
```

<XM291rb.m>

```
1  % XM291rb.m -> A working script M-file to execute analysis of an XM291, with
2  % one rigid body mode, using the finite element formulation and dynamics
3  % analysis functions defined within the subdirectory, beam_fem/.
4  % o  Created 24 October 1995 - 08 January 1996 by Eric Kathe.
5  %  Benet Labs, Watervliet Arsenal, NY 12189-4050 <ekathe@pica.army.mil>
6  % ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
7
8  % ____________________________________________________________
9  % Section (1)
10 % Set the plot and print flags on or off, run the geometry
11 % m-file, and create a plot label.
12 % DEFINE: plot_on, print_on, tlabel, spatial, lden, lEI, lnbden, Mextl,
13 %       Mextr, gm.
14 % ..........................................................
15 %
16 % Set plot and print flags to 1 to enable, zero to disable.
17 plot_on = 1;
18 print_on = 0;
19 %
20 if print_on == 1
21  plot_on = 1;  % Clearly, to print, the plot flag must be enabled.
22 end
23 %
24 tlabel = 'XM291rb';% tlabel -> Enables plotting.
25 %
26 if plot_on == 1
27  figure(1)
28  clg
29  % Define the figure options for desired output.
30  set(gcf,'PaperOrientation','portrait');  % This series of commands configures
31  set(gcf,'PaperUnits','inches');          % the plot window to effectively
32  set(gcf,'PaperPosition',[1 1 6.5 4.5]);  % be incorporated into a report.
33  set(gcf,'Units','inches','position',get(gcf,'PaperPosition'));
34  set(gcf,'DefaultAxesFontSize',9);
35  %
36  [spatial, lden, lEI, lnbden, Mextl, Mextr, gm] = geomf_XM291(tlabel);
37 else
38  [spatial, lden, lEI, lnbden, Mextl, Mextr, gm] = geomf_XM291;
39 end
40 %  spatial -> Axial position vector.
41 %  lden -> Beam linear density vector.
42 %  lEI -> Similar to lden except for linear EI cross-section properties.
43 %  lnbden -> The inertia of non-beam masses attached to the beam.
44 %  Mextl, Mextr -> 2x2 sub matrices of left and right extreme rigid body
45 %             inertia.
46 %  gm -> The columns of this matrix record the inner and outer radii.
47 %
48 if print_on == 1
49  print -deps fig.ps; % Print the file as an encapsulated Post-Script file.
50 end
51 %
52 % ____________________________________________________________
53 % Section (2)
54 % Define barrel constraint locations, number of elements, and create
55 % element mesh vector.
56 % DEFINE: snlv, ncord.
57 % USE: spatial, lden, lEI, lnbden, snlv, tlabel.
58 % ..........................................................
59 %
60 elmechloc = 0.220;% (m) Mid-span of breech ring threads on barrel.
61 trunnionloc = 38.9*(0.0254);% in(m/in) Yoke position as in geomf_XM291
62 %                          line 462.
63 snlv = [elmechloc; trunnionloc];% (m) Imposed node location vector.
64 %
65 nel = 7;% A relatively small number of FEM elements to keep the
66 %       problem quick & manageable for initial controls work.
67 %
68 if plot_on == 1
69  figure(2)
70  clg
71  % Define the figure options for desired output.
72  set(gcf,'PaperOrientation','portrait');  % This series of commands configures
73  set(gcf,'PaperUnits','inches');          % the plot window to effectively
74  set(gcf,'PaperPosition',[1 1 4 4.5]);    % be incorporated into a report.
75  set(gcf,'Units','inches','position',get(gcf,'PaperPosition'));
76  set(gcf,'DefaultAxesFontSize',9);
77  %
78  [ncord] = fem_mesh(spatial,lden,lEI,lnbden,snlv,nel,tlabel);
79 else
80  [ncord] = fem_mesh(spatial,lden,lEI,lnbden,snlv,nel);
81 end
82 % <ncord> is the vector of node indices.
83 %
84 if print_on == 1
85  print -deps fig2.ps; % Print the file as an encapsulated Post-Script file.
86 end
87 %
88 elmechloc = []; trunnionloc = []; nel = [];
89 %
90 % ____________________________________________________________
91 % Section (3)
92 % Generate the mass and stiffness matrices via FEM.
93 % DEFINE: Mfem, Kfem.
```

```
% USE: spatial, lden, lEI, lnbden, ncord, tlabel
% .....
%
[Mfem,Kfem] = fem_form(spatial,lden,lEI,lnbden,ncord);% Mfem & Kfem are
% out-put mass and stiffness matrices of the second-order self-adjoint
% system. Note tlabel is not included so that a plot is not presented.
% _____
% Section (4)
% Generate the constraint matrix for elevation control problem. (This will
% just be the trunnion.) Also predefine the alpha and beta for Rayleigh
% damping. (Shames & Dyn, pp. 646.)
% DEFINE: alpha, beta,constraintm.
% USE: author supplied data.
% .....
%
alpha = 0.000; % Mass proportional damping factor.
beta = 0.001; % Stiffness proportional damping factor.
%
kcnst = 750000*(4.4482)/(0.0254);% (lbf/in)(N/lbf)/(m/in)
% Estimated lateral elasticity of the Trunnion constraint.
%
cdcnst = beta*kcnst; % Stiffness proportional damping of constraint.
%
[y, k1tr] = min(abs(snlv(2) - spatial(ncord)));% Identification of
%                                   constraint node number.
gcindk = 2*[k1tr] - 1;% Identification of constraint generalized
%                   coordinate number.
%
constraintm = [gcindk kcnst cdcnst];
%
kcnst = []; cdcnst = []; y = []; k1tr = []; gcindk = [];
% _____
% Section (5)
% Lump in the external masses and constraint into the Mass and Stiffness
% matrices. Also generate the Rayleigh damping matrix.
% DEFINE: M, K, Cd, n2.
% USE: Mfem, Kfem, Mextl, Mextr, alpha, beta, constraintm.
% .....
%
[M,K,Cd] = fem_lump(Mfem,Kfem,Mextl,Mextr,alpha,beta,constraintm);
%   M, K, Cd -> Mass, stiffness, and Rayleigh damping matrices of
%        generalized coordinates that include the constraint and
%        externally coupled dynamics.
%
n2 = size(M,1);% The number of generalized coordinates.
%
if plot_on == 1
 figure(3);
 clg
 % Define the figure options for desired output.
 set(gcf,'PaperOrientation','portrait'); % This series of commands configures
 set(gcf,'PaperUnits','inches');        % the plot window to effectively
 set(gcf,'PaperPosition',[1 1 6.5 4.5]); % be incorporated into a report.
 set(gcf,'Units','inches','position',get(gcf,'PaperPosition'));
 set(gcf,'DefaultAxesFontSize',9);
 %
 colormap(flipud(hot));% Assign non-default color map for better gray scale.
 subplot(321), image(64*M/max(max(M))); % Normalize M to 0 to 64.
 title(['Image of Mass Matrix, M'])
 set(gca,'aspect',[1,1]); % This aspect ratio makes the matrix images square.
 subplot(11,2,11), image(1:64); % This plots the color bar or image scale.
 title('Color Scale')
 xlabel = ['Zero';'High'];
 set(gca,'XTick',[1,64])
 set(gca,'XTickLabels',xlabel)
 set(gca,'YTick',[])
 subplot(325), image(64*K/max(max(K))) % Normalize K to 0 to 64.
 title(['Image of Stiffness Matrix, K'])
 set(gca,'aspect',[1,1])
 subplot(222), spy(M)
 title('Spy Image of Nonzero Mass Matrix Elements')
 subplot(224), spy(K)
 title('Spy Image of Nonzero Stiffness Matrix Elements')
 subtitle([tlabel ' Constrained Mass & Stiffness Matrices'])
end
%
if print_on == 1
 print -deps fig3.ps; % Print the file as an encapsulated Post-Script file.
end
%
% _____
% Section (6)
% Generate the second order, undamped eigen-modes.
% DEFINE: phi, fvn, rlab.
% USE: M, K.
% .....
%
[phi,fvn,rlab] = eigen_2o(M,K);% Compute, normalize, sort, and identify
%                   the modes of vibration.
%   phi -> n2xn2 matrix whose columns consist of the mass normalized eigen-
%        vectors with each column corresponding the respective frequency
%        element of fv.
%   fvn -> n2x1 vector of linear modal frequencies (Hz) sorted in order of
%        increasing frequency.
%   rlab -> String matrix of row labels identifying the mode's by
%        bending mode number or Rigid body mode.
if plot_on == 1
 figure(4)
 clg
 % Define the figure options for desired output.
 set(gcf,'PaperOrientation','portrait'); % This series of commands configures
 set(gcf,'PaperUnits','inches');        % the plot window to effectively
 set(gcf,'PaperPosition',[1 1 6.5 4.5]); % be incorporated into a report.
 set(gcf,'Units','inches','position',get(gcf,'PaperPosition'));
 set(gcf,'DefaultAxesFontSize',9);
 n = 0;
 for j = 1:3
  for k = 0:3:3
   l = j+k;
   n = n+1;
   subplot(3,2,n)
   beam_plot(ncord,spatial,phi(:,l),gm/5,constraintm(:,1));
   mlabel = rlab(i,find(abs(rlab(l,:)) ~= 32));
   if strcmp(rlab(i,1),'B') % Check to see if bending mode.
    title([mlabel ', ' num2str(fvn(l)) 'Hz'])
   else % Else -> Rigid body with near-zero frequency not displayed.
    title([mlabel])
   end
   axis([-0.2 7 -.1 .1])
   grid
   if n > 4
    xlabel('Axial Position (m)')
   end
   pos = get(gca,'position'); % This is in normalized coordinates
   pos(4)=pos(4)*.85;       % Shrink the height by a factor of .85
   pos(2)=pos(2) + pos(4)*.15;% Raise the subplot by the saved height.
   set(gca,'position',pos);
  end
 end
 subtitle([tlabel ' Undamped Mass-Normalized Eigenvectors and Frequencies'])
end
%
if print_on == 1
 print -deps fig4.ps; % Print the file as an encapsulated Post-Script file.
end
%
n = []; k = []; l = []; mlabel = [];
%
% _____
% Section (7)
% Generate the second order, damped eigen-modes.
% DEFINE: phi, fv, rlab.
% USE: M, K, Cd.
% .....
%
[phi,fv,rlab] = eigen_2o(M,K,Cd);% Compute, normalize, sort, and identify
%                   the modes of vibration.
%   phi -> n2xn2 matrix whose columns consist of the mass normalized eigen-
%        vectors with each column corresponding the respective frequency
%        element of fv.
%   fv -> n2x1 vector of linear modal frequencies (Hz) sorted in order of
%        increasing frequency.
%   rlab -> String matrix of row labels identifying the mode's by
%        bending mode number or Rigid body mode.
if plot_on == 1
 figure(5)
 clg
 % Define the figure options for desired output.
 set(gcf,'PaperOrientation','portrait'); % This series of commands configures
 set(gcf,'PaperUnits','inches');        % the plot window to effectively
 set(gcf,'PaperPosition',[1 1 6.5 4.5]); % be incorporated into a report.
 set(gcf,'Units','inches','position',get(gcf,'PaperPosition'));
 set(gcf,'DefaultAxesFontSize',9);
 n = 0;
 for j = 1:3
  for k = 0:3:3
   l = j+k+1;
   n = n+1; % By adding one, the rigid body mode is skipped.
   subplot(3,2,n)
   beam_plot(ncord,spatial,phi(:,l),gm/5,constraintm(:,1));
   mlabel = rlab(i,find(abs(rlab(l,:)) ~= 32));
   if strcmp(rlab(i,1),'B') % Check to see if bending mode.
```

```
      title([rmlabel ', num2str(fvn(i)) 'Hz'])
    else % Else -> Rigid body with near-zero frequency not displayed.
      title([rmlabel])
    end
    axis([-0.2 7 -.1 .1])
    grid
    if n > 4
      xlabel('Axial Position (m)')
    end
    pos = get(gca,'position'); % This is in normalized coordinates
    pos(4)=pos(4)*.85;        % Shrink the height by a factor of .85
    pos(2)=pos(2) + pos(4)*.15;% Raise the subplot by the saved height.
    set(gca,'position',pos);
  end
end
subtitle([tlabel ' Damped Mass-Normalized Bending Mode-Shapes and Frequencies'])
end
%
if print_on == 1
  print -deps fig5.ps; % Print the file as an encapsulated Post-Script file.
end
%
n = []; k = []; l = []; rmlabel = [];
%
% _____________________________________________________________________
% Section (8)
% Convert to first-order state-space and truncate to muzzle, elmech SISO.
% DEFINE: A, B, C, D, nq, states, outputs, inputs, ae, be, ce, de.
% USE: M, K, Cd.
% ..........................................................
%
[A,B,C,D] = fem2ss(M,K,Cd);% This form assumes F1 through Mn inputs
%                          and all generalized coordinates as outputs.
%   A, B, C, D -> State-Space Matrices as in:
%   dqv/dt = A*qv + B*uv
%   yv  = C*qv + D*uv ... where qv = state vector ([gc's; dgc's/dt]).
%                         uv = input vector F(1) through M(n2/2).
%                         yv = output of generalized coords.
nq = size(A,1);% The number of state variables.
%
if plot_on == 1
  figure(6)
  clg
  set(gcf,'PaperOrientation','portrait');  % This series of commands configures
  set(gcf,'PaperUnits','inches');          % the plot window to effectively
  set(gcf,'PaperPosition',[1 1 4 4]);  % be incorporated into a report.
  set(gcf,'Units','inches','position',get(gcf,'PaperPosition'));
  set(gcf,'DefaultAxesFontSize',9);
  subplot(221), spy(A)
  title('A matrix')
  subplot(222), spy(B)
  title('B matrix')
  subplot(223), spy©
  title('C matrix')
  subplot(224), spy(D)
    pos = get(gca,'position');  % This is in normalized coordinates
    pos(4)=pos(4)/2;          % Shrink the height by a factor of .85
    pos(2)=pos(2) + pos(4)/2; % Raise the subplot but keep it centered.
    set(gca,'position',pos);
  title('D matrix')
  subtitle(['First-Order, Non-Zero, System Matrix Entries'])
end
%
if print_on == 1
  print -deps fig6.ps; % Print the file as an encapsulated Post-Script file.
end
%
% To print the system using the printsys command, uncomment this subsection.
% ^^^^^^^^^^
  % gcs = [];   % Declare null initial variable description strings.
  % dgcs = [];  %
  % inputs = []; %
  % for I = 1:n2/2
  %   gcs = [gcs ' x' int2str(I) ' theta' int2str(I)];   % Loop through &
  %   dgcs = [dgcs ' dx' int2str(I) ' dtheta' int2str(I)];% define each
  %   inputs = [inputs ' F' int2str(I) ' M' int2str(I)];  % beam variable.
  % end
  % states = [gcs dgcs];% State vector ([gc's; dgc's/dt])
  % outputs = [gcs];   % Output of generalized coords.
  %
  % inputs(1) = [];   %
  % states(1) = [];   % Eliminate initial blanks.
  % outputs(1) = [];  %
  %
  % printsys(A,B,C,D,inputs,outputs,states)
% ^^^^^^^^^^
%
% Select system for elevation mechanism input, F2 -> gc3, and
% muzzle pointing angle output, theta(nel+1) -> final gc -> gc(n2).
%
[ae,be,ce,de] = sselect(A,B,C,D,3,n2);
%
% printsys(ae,be,ce,de,'F-elmech','Muzzle_Pointing_Angle',states)
%
% _____________________________________________________________________
% Section (9)
% Pole Zero Map.
% USE: ae, be, ce, de, fvn.
% ..........................................................
%
[p,z] = pzmap(ae,be,ce,de);
wv = imag(p(find(imag(p) > .1)));% Find imaginary part of positive conjugates.
wv = [0; sort(wv(:))];
%
if plot_on
  figure(7)
  clg
  set(gcf,'PaperOrientation','portrait');  % This series of commands configures
  set(gcf,'PaperUnits','inches');          % the plot window to effectively
  set(gcf,'PaperPosition',1.4*[1 1 4 4*4.4/3.3]);    % be incorporated into a report.
  set(gcf,'Units','inches','position',get(gcf,'PaperPosition'));
  set(gcf,'DefaultAxesFontSize',9);
  whitebg(gcf,'k')
  pzmap(ae,be,ce,de)
  title([tlabel ' Open-Loop Eigenvalues'])
  axis([-2.2*max(wv) 1.1*max(wv) -2.2*max(wv) 2.2*max(wv)])
  flabel = [];
  for I = 1:length(wv);
    flabel = str2mat(flabel, freq2str(wv(I)/(2*pi),1));
  end
  flabel(1,:) = [];
  set(gca,'YTick',wv)
  set(gca,'YTickLabels',flabel)
  set(gca,'YGrid','on')
  % Use natural frequency (undamped), fvn, for sgrid command.
  waxism = sort(fvn(find(real(fvn) < 2.2*max(wv)/(2*pi)))*2*pi);
  waxism(1) = [];
  sgrid(.2:.2:1,waxism)
  ylabel('Imaginary Axis,     Damped Frequencies')
  xlabel('Real Axis, Exponential Decay Rate (1/sec)')
  hold on
  % Mask out horizontal grid lines to the right of the poles.
  for I = 1:length(wv);
    if wv(I) > .1
      plot([real(p(find(imag(p) == wv(I)))) 1.09*max(wv)], wv(I)*[1 1],'k')
    end
  end
  % Replot the Imaginary axis.
  plot([0 0],2.2*max(wv)*[-1 1],'w:')
  % Plot natural frequency manual grid-lines to the left.
  for I = 1:length(waxism)
    plot([1 0.5*max(wv)],waxism*[1 1],'w:')
  end
  % ylabel('Imaginary Axis')
  set(gca,'AspectRatio',[3.3/4.4 1])
  pos = get(gca,'position');  % This is in normalized coordinates
  pos(1)= pos(1) + 0.05*pos(3);% + pos(3)*0.10; % Raise the subplot but keep it centered.
  set(gca,'position',pos);
  nfpos = pos;
  nfpos(1) = pos(1) + pos(3);
  nfpos(4) = 0.95*nfpos(4);
  nfpos(2) = pos(2) + 0.025*pos(4);
  axes('position', nfpos)
  axis([-2.2*max(wv) 1.1*max(wv) -2.2*max(wv) 2.2*max(wv)])
  set(gca,'XColor', 'k')
  set(gca,'XColor', 'k')
  flabel = [];
  for I = 1:length(waxism);
    flabel = str2mat(flabel, freq2str(real(waxism(I)/(2*pi)),1));
  end
  flabel(1,:) = [];
  set(gca,'YTick',waxism)
  set(gca,'YTickLabels',flabel)
  ylabel(['                                    ' ...
          '                                    ' ...
          'Undamped Natural Frequencies'])
  hold off
end
%
if print_on == 1
  set(gcf,'InvertHardCopy','on')
  print -deps fig7.ps
```

```
end
% _____
% Section (10)
% Bode Plot.
% USE: ae, be, ce, de.
% .....
%
%
[mag,phase,w] = bode(ae,be,ce,de,1,logspace(log10(5*2*pi),log10(500*2*pi),500));
if plot_on == 1
 figure(8)
 clg
 set(gcf,'PaperOrientation','portrait');  % This series of commands configures
 set(gcf,'PaperUnits','inches');          % the plot window to effectively
 set(gcf,'PaperPosition',[1 1 6.3 3]);    % be incorporated into a report.
 set(gcf,'Units','inches','position',get(gcf,'PaperPosition'));
 set(gcf,'DefaultAxesFontSize',9);
 whitebg(gcf,'k')
 subplot(211), semilogx(w/(2*pi),20*log10(mag))
 title([flabel ' Elevation Mechanism to Muzzle Pointing Angle Bode Plot'])
 ylabel('Gain dB')
 av = [ [min(w) max(w)]/(2*pi) min(20*log10(mag)) 0.95*max(20*log10(mag))];
 axis(av)
 fabsissa = sort([5; wv(2:5)/(2*pi); 500]);
 flabel = [];
 for I = 1:length(fabsissa)
  flabel = str2mat(flabel,freq2str(fabsissa(I),1));
 end
 flabel(1,:) = [];
 set(gca,'XTick',fabsissa)
 set(gca,'XTickLabels',flabel)
 set(gca,'YTick',[min(20*log10(mag)), median(20*log10(mag)), max(20*log10(mag))])
 grid
 subplot(212), semilogx(w/(2*pi),phase)
 xlabel('Frequency (Hz)'), ylabel('Phase deg')
 av = [ [min(w) max(w)]/(2*pi) min(phase) (max(phase)+180)];
 axis(av)
 set(gca,'YTick',[180:-180:-600])
 grid
end
%
if print_on == 1
 set(gcf,'InvertHardCopy','on')
 print -deps fig8.ps; % Print the file as an encapsulated Post-Script file.
end
%
% _____
% Section (11)
% Unit Impulse Response.
% USE: ae, be, ce, de.
% .....
%
%
[y,x,t] = impulse(ae,be,ce,de,1,[0:0.00001:0.2]);
%
if plot_on == 1
 figure(9)
 clg
 set(gcf,'PaperOrientation','portrait');  % This series of commands configures
 set(gcf,'PaperUnits','inches');          % the plot window to effectively
 set(gcf,'PaperPosition',[1 1 3.8 3.5]);  % be incorporated into a report.
 set(gcf,'Units','inches','position',get(gcf,'PaperPosition'));
 set(gcf,'DefaultAxesFontSize',9);
 plot(t,y,'k',[min(t) max(t)],[0 0],'k:')
 av = axis;
 av(3) = 2.2*av(3);
 axis(av)
 pos = get(gca,'position');          % This is in normalized coordinates
 pos(2)= pos(2) + (1-0.85)*pos(4)/2; % Raise the subplot but keep it centered.
 pos(4)= 0.85*pos(4);                % Shrink the height by a factor of .85
 set(gca,'position',pos);
 title('SISO Unit Impulse Response')
 ylabel('Muzzle Deflection (rad)')
 xlabel('Time After Newton-Second Impulse at Elevation Mechanism (sec)')
 newpos = [(pos(1)+pos(3)/6) (pos(2)+pos(4)/7) (pos(3)/2) (pos(4)/3)];
 axes('position',newpos)
 ttrunk = t(1:(length(t)/30));
 plot(ttrunk,y(1:length(ttrunk)),'k',[0 10],[0 0],'k:')
 av = axis;
 av([3 4]) = av([3 4])/3;
 av([1 2]) = [0 0.010];
 axis(av)
 title('                Initial Response Close-Up')
 ylabel('(rad)')
 set(gca,'XTick',[0 0.005 0.010])
 set(gca,'XTickLabels',str2mat('0.000','0.005','0.010'))
 xlabel('(sec)')
end
%
if print_on == 1
 print -deps fig9.ps; % Print the file as an encapsulated Post-Script file.
end
%
av = []; pos = []; ttrunk = [];
% _____
% Completed:
% .....
```

<beam_plot.m>

```
function [] = beam_plot(ncord,spatial,y,sgm,constraintgc,extgc)
% beam_plot.m
% [] = beam_plot(ncord,spatial,y,sgm,extgc);
% o  This m-file generates a plot of the output state of a beam system.
% o  This file uses:
%      fem_node_check.m in section 1.
%       geom_check.m
%      fem_interp.m in section 2.
% o  External generalized coordinates may be include via extgc.
% o  Specific input/output variable definitions are:
%   ncord -> The vector of indices such that spatial(ncord) is the position
%     of each pair of beam generalized coordinates. (Also known as nodes.)
%   spatial -> Axial position vector, currently limited to equally spaced
%     position vectors with position data of every dx*n point such that
%     n = 1:length(spatial) and length(spatial)*dx equals max(spatial).
%   y -> Output vector. Contains the magnitudes of the displacements and
%     rotations of the generalized coordinates.  It is a vector of the form:
%     [x(1) theta(1) x(2)...x(n2) theta(n2) extgc(1) extgc(2)...extgc(nxgc)]'.
%   sgm -> sgm, scaled geometry matrix to describe the beam.
%   constraintgc -> Inertial constraint generalized coordinate vector.
%         Contains the index of the generalized coordinate that is
%         coupled to the ground inertial reference frame.  Use [] in
%         the event of free-free.
%   extgc -> External generalized coordinate vector.  Contains the index of
%         the generalized coordinate to which it is coupled.
% o  Created 3 October - 1995 12 January 1996 by Eric Kathe.
%   Benet Labs, Watervliet Arsenal, NY  12189-4050 <ekathe@pica.army.mil>
% Version 1
% ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

% _____
% Section (1)
% A few checks to be sure input variables are the right size, et cetera.
% This section uses geom_check.m to validate the input geometry vectors.
% Also define the number of indices of spatial, ns, the number of generalized
% coordinates ngc, et cetera.
% DEFINE: nel, ngc, nn, ns, nxgc.
% USE: extgc, constraintgc, nargin, ncord, spatial, y,
%     author provided default value.
% POSSIBLY ALTER: constraintgc, extgc, ncord, spatial, y.
% .....
% (A) Check spatial & ncord & define their length's, ns.
%
[ncord,spatial] = fem_node_check(ncord,spatial);
ns = length(spatial); % Number of indices of spatial.
nn = length(ncord); % Number of nodes of ncord.
nel = nn - 1;% Number of elements of the beam.
%
%
% (B) Check  & modify inertial constraint vector.
%
if constraintgc ~= [];
 constraintgc = constraintgc(:);
 constraintgc = sort(constraintgc);
 if sum(ceil(constraintgc) - constraintgc) > 0
  constraintgc = round(constraintgc);
  warning = 'Constraintgc must be of integer values.  It has been rounded.'
 end
 if min(constraintgc) < 1
  constraintgc = [];
  warning = ['Constraintgc values must match a generalized coordinate number. '...
        'Negative values inconstant.  Constraintgc has been nullified.']
 end
 if max(constraintgc) > 2*nn
  constraintgc = [];
  warning = ['Constraintgc values must match a generalized coordinate number. '...
        'Values that exceed the number of generalized coordinates '...
        ' are inconstant.  Constraintgc has been nullified.']
 end
end
%
% © Check to see if extgc included.  If so, check it
% & define its length, nxgc; else set nxgc = 0.
```

```
74 %
75 nxgc = 0;
76 if nargin == 6
77   extgc = extgc(:);
78   extgc = sort(extgc);
79   if sum(ceil(extgc) - extgc) > 0
80     extgc = round(extgc);
81     warning = 'Extgc must be of integer values.  It has been rounded.'
82   end
83   if min(extgc) < 1
84     extgc = [];
85     warning = ['Extgc values must match a generalized coordinate number.  '...
86           'Negative values inconstant.  Extgc has been nullified.']
87   end
88   if max(extgc) > 2*nn
89     extgc = [];
90     warning = ['Extgc values must match a generalized coordinate number.  '...
91           'Values that exceed the number of generalized coordinates '...
92           ' are inconstant.  Extgc has been nullified.']
93   end
94   nxgc = length(extgc);
95 end
96 %
97 % (D) Impose column structure on y, and check constancy of number of
98 % generalized coordinates.
99 %
100 y = y(:);
101 ngc = length(y);
102 if ngc ~= (2*nn + nxgc)
103   if ngc < 2*nn
104     warning = ['More generalized coordinates than the number of outputs.'...
105           'No corrective action has been taken.']
106   elseif nargin == 5
107     zgext = ones((ngc - 2*nn),1);
108     zgext(1:nxgc) = extgc;
109     extgc = zgext(1:(ngc - 2*nn));
110     warning = ['Mismatched number of generalized & or external '...
111           'couplings.  extgc padded or truncated to fit.  '...
112           'Further errors are very likely.']
113   else
114     warning = ['Fewer generalized coordinates than the number of outputs.'...
115           'No corrective action has been taken.']
116   end
117 end
118 %
119 default = []; warning = [];
120 % ____________________________________________________________________
121 % Section (2)
122 % Compute the beam deflection vectors for each finite element using
123 % fem_interp.m, and combine into one beam deflection vector.
124 % DEFINE: deflectv.
125 % USE: ncord, nn, ns, spatial, y,
126 % ....................................................................
127 %
128 % (A) Initialize deflection vector.
129 %
130 deflectv = zeros(size(spatial));
131 %
132 % (B) For the first element of the beam.
133 %
134 lengvseg = spatial(ncord(1):ncord(2)); % Length of finite element.
135 [zphi, zddphi] = fem_interp(lengvseg);% Zphi contains the shape functions.
136 %
137 xv = y(1:4);% The deflection of element one is a function of the
138 %          state of both adjacent nodes. In this case
139 %          {x1, theta1, x2, theta2}.
140 %
141 deflectv(ncord(1):ncord(2)) = zphi*xv;
142 %
143 % © For remaining elements:
144 %
145 for I=2:nel;
146   %
147   lengvseg = spatial((ncord(I)+1):ncord(I+1)) - spatial(ncord(I));
148   [zphi, zddphi] = fem_interp(lengvseg);
149   indL = 2*I - 1;
150   indR = indL + 3;
151   deflectv((ncord(I)+1):ncord(I+1)) = zphi*y(indL:indR);
152 end
153 %
154 I = []; indL = []; indR = []; xv = []; lengvseg = []; zphi = [];
155 zddphi = [];
156 % ____________________________________________________________________
157 % Section (3)
158 % Plot the deflected geometry, and external general coordinates in
159 % the current figure window.  Note: labels and axis may be set after
160 % the function call.  Also note that their is room for improvement with
161 % rotational external absorbers.
162 % USE: spatial, deflectv, ncord, nxgc sgm, smperk, y, extgc, constraintgc.
163 % ....................................................................
164 %
165 % (A) Compute & plot the plotting geometry matrix. It is composed of the
166 %     scaled geometry matrix and its reflection about the axis, offset
167 %     by the deflection vector computed in section 2.  Hold the plot.
168 %
169 pgm = [sgm -sgm] + deflectv*ones(1,2*size(sgm,2));
170 plot(spatial,pgm,'k')
171 hold on
172 %
173 % (B) Include a centerline and mark off the deflected node locations.
174 %
175 plot(spatial,deflectv,'b:')
176 plot(spatial(ncord),deflectv(ncord),'kx')
177 %
178 % © Plot inertial constraint locations separately.
179 %
180 if length(constraintgc) > 0
181   for I = 1:length(constraintgc)
182     xspat = spatial(ncord(ceil(constraintgc(I)/2)));
183     xdeflect = [0; deflectv(ncord(ceil(constraintgc(I)/2)))];
184     if ((constraintgc(I)/2) - floor(constraintgc(I)/2)) > 0;
185       % Odd g.c. is translation.
186       lwidth = get(gca,'LineWidth');
187       % Plot a wide line connection:
188       line(xspat*[1 1],xdeflect,'LineWidth',5*lwidth,'Color','b')
189       % plot(xspat*[1 1],xdeflect,'k-')
190       plot(xspat,xdeflect(2),'ko')
191     else
192       plot(xspat*[1 1],xdeflect,'g-')
193       plot(xspat,xdeflect(2),'g*')
194     end
195   end
196 end
197 %
198 % (D) If external absorbers are included, plot them separately.
199 %
200 if nargin == 6
201   for I = 1:nxgc
202     xspat = spatial(ncord(ceil(extgc(I)/2)));
203     xdeflect = [y(2*nn+I); deflectv(ncord(ceil(extgc(I)/2)))];
204     if ((extgc(I)/2) - floor(extgc(I)/2)) > 0; % Odd g.c. is translation.
205       plot(xspat*[1 1],xdeflect,'r')
206       plot(xspat,xdeflect(1),'ro')
207     else
208       plot(xspat*[1 1],xdeflect,'g')
209       plot(xspat,xdeflect(1),'g*')
210     end
211   end
212 end
213 %
214 hold off
215 % ____________________________________________________________________
216 % Completed:
217 % OUTPUT: None.
218 % ....................................................................
219 %
```

<eigen_2o.m>

```
1  function [phi,fv,rlab] = eigen_2o(M,K,Cd)
2  % eigen_2o.
3  % [phi,fv,rlab] = eigen_2o(M,K,Cd);
4  % o  This m-file computes the modal matrix phi whose columns consist of
5  %    eigen vectors in order of increasing corresponding frequency, omega.
6  % o  The eigen vectors are mass normalized.
7  % o  The relationships between upper and lower halves of the first-order
8  %    eigenvectors, used for damped mode shape determination, is discussed
9  %    in a commented portion of section four of this m-file.
10 % o  This file uses:
11 %    M, K -> n2xn2 mass and stiffness matrices where n2 = number
12 %         of generalized coordinates.
13 %    Cd   -> Optional n2xn2 damping matrix where n2 = number
14 %         of generalized coordinates.
15 %    phi -> n2xn2 matrix whose columns consist of the mass normalized eigen-
16 %         vectors with each column corresponding the respective frequency
17 %         element of omega.
18 %    fv -> n2x1 vector of linear modal frequencies (Hz) sorted in order of
19 %         increasing frequency.
20 %    rlab -> Optional string matrix of row labels identifying the mode's by
21 %         bending mode number or Rigid body mode.
22 % o  Created 16 October - 28 November 1995 by Eric Kathe.
23 %    Benet Labs, Watervliet Arsenal, NY  12189-4050 <ekathe@pica.army.mil>
24 %    Version 1.
```

```
% ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
%
% _____________________________________________________________________
% Section (1)
% Check the mass and stiffness matrices are real and square, the
% same size, and symmetric.
% DEFINE: n2.
% USE: M, K.
% POSSIBLY ALTER: M, K.
% ....................................................................
%
[mm,nm] = size(M);
[mk,nk] = size(K);
%
n2 = min([mm nm mk nk]);% The number of generalized coordinates.
%
if mm ~= nm | mk ~=nk | mm ~= mk
 M = M(1:n2,1:n2);
 K = K(1:n2,1:n2);
 warning = ['Mass and stiffness matrices must be same size and ' ...
        'square. They have been truncated to ' int2str(n2) ...
        ' elements square.']
end
%
if ~isreal(M)
 M = abs(M)
 warning = 'Mass matrix must be real. It has been set to abs(M).'
end
if ~isreal(K)
 K = abs(K)
 warning = 'Stiffness matrix must be real. It has been set to abs(K).'
end
%
if sum(sum(M ~= M')) ~= 0
 warning = 'M not symmetric, errors are possible.'
end
if sum(sum(K ~= K')) ~= 0
 warning = 'K not symmetric, errors are possible.'
end
%
mm = []; nm = []; mk = []; nk = []; warning = [];
% _____________________________________________________________________
% Section (2)
% Check if the damping matrix has been included. If so, check it as was
% done for the mass and stiffness matrices in section 1.
% DEFINE: damp_flag.
% USE: Cd, n2.
% POSSIBLY ALTER: Cd.
% ....................................................................
%
damp_flag = 0; % Initialize the damping flag to off.
%
if nargin == 3 % Check to see if the damping matrix was included.
 damp_flag = 1;
 [mc,nc] = size(Cd);
 %
 if mc ~= nc          % Check to see if the damping matrix is square
  nc = min([mc nc]); % If it isn't square, truncate it to be square
  Cd = Cd(1:nc,1:nc); % and warn of the problem.
  warning = 'Damping matrix was not square. It has been truncated.'
 end
 %
 if nc < n2           % Check to see if the damping matrix is the
  Zd = zeros(n2,n2);  % same size as M and K. If not, pad or
  Zd(1:nc,1:nc) = Cd; % or truncate to the same size and warn of the
  Cd = Zd;            % problem.
  warning = ['Damping matrix size smaller than M & K. '...
        'It has been zero padded to match the size.']
 elseif nc > n2
  Cd = Cd(1:n2,1:n2);
  warning = ['Damping matrix size is larger than M & K. '...
        'It has been truncated to match the size.']
 end
 %
 if ~isreal(Cd)
  Cd = abs(Cd)
  warning = 'Damping matrix must be real. It has been set to abs(Cd).'
 end
 %
 if sum(sum(Cd ~= Cd')) ~= 0
  warning = 'Cd not symmetric, errors are likely.'
 end
 mc = []; nc = []; Zd = []; warning = [];
 %
end
% _____________________________________________________________________
% Section (3)
% Compute the undamped eigen system.
% M*ddx + K*x = 0
% (w^2*M + K)*x = 0
% K*x = w^2*M*x
% A*x = lambda*B*x    The generalized eigenvalue problem.
% DEFINE: phi, wv.
% USE: M, K.
% ....................................................................
[phi,lambda] = eig(K,M);
%
wv = sqrt(diag(lambda));% Circular frequency is the square root of lambda.
%
lambda = [];
% _____________________________________________________________________
% Section (4)
% If damping is included:
% Compute the symmetric first-order symmetric state equations,
% Find the first-order eigen values and vectors,
% Extract the relevant eigen values and eigen vectors, and
% convert the eigen values to radial frequency.
% DEFINE: phi, lambda.
% USE: M, K, Author supplied cutoff frequency.
% ....................................................................
if damp_flag == 1
 %
 % Identify the expected number of rigid body modes:
 %
 nrb = size(K,1) - rank(K);  % It is well know that the number of rigid
 %                           body modes of a beam matches the rank
 %                           deficiency of the stiffness matrix.
 %
 % Identify the fundamental, undamped, flexible mode frequency:
 %
 wv = sort(wv); % Sort the undamped frequencies.
 fmwv = wv(nrb + 1); % The fundamental flexible mode frequency is the
 %                   lowest frequency following the rigid body 'zero'
 %                   frequencies.
 %
 % Define a cut-off frequency below which the
 % mode is over damped or rigid body within numeric errors.
 % 5% of the undamped fundamental seems a reasonable
 % cut-off. (Note: if heavy use of mass proportional damping is
 % employed, low frequencies may be overdamped, and confuse this
 % scheme. Generally, mass proportional damping is small relative
 % to the stiffness damping, reducing the potential for problems.)
 %
 fcut = fmwv/20;
 %
 % Symmetric homogenous state equations: At qdot + Bt q = 0
 %
 % [ 0  M ][ xddot ]  [ -M  0 ][ xdot ]  [ 0 ]
 % [ M  Cd ][ xdot ] + [ 0  K ][  x  ] = [ 0 ]
 %
 At = [zeros(n2,n2) M; M Cd];
 Bt = [-M zeros(n2,n2); zeros(n2,n2) K];
 %
 % First order generalized eigen problem:
 % lambda At q  + Bt q = 0 -> -Bt q = lambda At q
 %        -   -     -     -
 %
 [ssphi,lambda] = eig(-Bt,At);
 %
 sslamv = diag(lambda); % Convert diagonal eigenvalue matrix to vector.
 %
 %
 % Identify flexible modes, by finding values with significant imaginary
 % components. Isolate only one half of the conjugate pairs by only
 % keeping the roots with positive imaginary components.
 %
 [indflex] = find( imag(sslamv) >= fcut );
 indflex = indflex(:);
 %
 % Identify rigid body modes by their lack of imaginary content, and their
 % close proximity to the origin.
 %
 [indrigid] = find( imag(sslamv) >= 0 & imag(sslamv) < fcut & ...
          abs(real(sslamv)) < 100*fcut );
 indrigid = indrigid(:);
 %
 if size(indrigid) ~= nrb
  warning = ['The number of rigid body eigenvalues found does not '...
        'match the rank deficiency of K. The eigenvector labels '...
        'may not properly indicate the rigid body modes.']
 end
 %
```

```
wv = imag(sslamv([indflex; indrigid]));   % Assemble radial frequencies and
phi = real(ssphi(1:n2,[indflex; indrigid]));% respective eigenvectors.
%
% NOTE: Matlab unit normalizes each eigenvector independently. Due to the
%       imaginary content of the eigen vectors, the normalization introduces
%       a new complex content to the eigenvectors. To investigate the
%       relationship between the upper and lower halves of the eigenvectors,
%       and the relationship between the eigenvectors of complex conjugate
%       eigenvalues, the complex normalization must first be removed. This
%       may be accomplished as follows:
% 1) Identify the complex conjugate pairs of under damped eigenvalues. This
%    may be done visually using: printmat([real(sslamv), imag(sslamv)]), or
%    it may be accomplished using:
%    cplxpair(sslamv,10^4*eps*max(abs(abs(sslamv)))). (Clearly, My experience
%    indicates that the numeric tolerance isn't very precise. Different
%    tolerances may be required if using the cplxpair command.)
% 2) Determine the complex normalization scale used. This may be done as
%    follows for the 12th and 13th eigenvectors as an example if
%    their eigenvalues are complex conjugates. (Substitute the indices of
%    the conjugate pairs identified in step one in lieu of 12 & 13.)
%    scale12 = ssphi(n2+1,12)./real(ssphi(n2+1,12))
%    scale13 = ssphi(n2+1,13)./real(ssphi(n2+1,13))
%    This uses the relationship that the lower half of the simplified eigen-
%    vectors will be real. The first such value is chosen for convenience.
%    (ssphi(n2+1:2*n2,12)./real(ssphi(n2+1:2*n2,12)) demonstrates each value.)
% 3) Renormalize the eigenvectors to impose the lower half of the vectors to
%    be real. Again, using the 12th & 13th eigenvectors as an example:
%    ev12 = ssphi(:,12)/scale12;
%    ev13 = ssphi(:,13)/scale13;
%    (Note the very small remaining imaginary content of the lower half.)
% 4) The desired relationship that the upper half of the eigen vectors equals
%    the lower half multiplied by the eigenvalue may be seen by comparing the
%    array division of the two halves with the eigenvalue:
%    ev12(1:n2)./ev12(n2+1:2*n2)
%    sslamv(12)
%    ev13(1:n2)./ev13(n2+1:2*n2)
%    sslamv(13)
% 5) NOTE that it is legitimate to take just the real part of the first half
%    of the eigen vector, and scale it later, as I have done. (All four
%    quadrants of the eigenvector -- upper/lower & imaginary/real -- differ
%    only by generally complex constants.)
%
feut = []; At = []; Bt = []; ssphi = []; lambda = []; sslamv = []; fmwv = [];
sslamreal = []; warning = []; inflex = []; indrigid = []; nrb = [];
%
end
% _____________________________________________________________________
% Section (5)
% Check and convert the eigen values to
% linear frequencies (Hz)and mass normalize the eigen vectors.
% Mass normalized -> phi'*M*phi = I.
% DEFINE: fv.
% USE: phi, lambda, ngc.
% Alter phi.
% .....................................................................
%
if ~isreal(wv)
  warning = ['Imaginary frequencies found.  System is likely to'...
             'be unstable.']
end
%
fv = wv/(2*pi);% Linear (Hz) frequency is the circular frequency/(2*pi).
%
norms = diag(phi'*M*phi);% For mass normalization, determine the norm-scale.
nphi = zeros(size(phi));
for I = 1:size(phi,2)
   nphi(:,I) = phi(:,I)/sqrt(norms(I));% Independently scale each vector.
   if nphi(2,i) < 0
     nphi(:,I) = - nphi(:,I); % Impose consistent sign convention.
   end                        % In this case.  Initial angle always up.
end
%
phi = nphi;
%
wv = []; norms = []; nphi = []; I = [];
% _____________________________________________________________________
% Section (4)
% Sort frequencies and eigen vectors in order of increasing frequency, and
% validate orthonormality of eigenvectors.
% USE: fv, phi.
% Alter fv, phi.
% .....................................................................
%
[y,i] = sort(fv);
fv = fv(I);
phi = phi(:,I);
%
% Test orthonormality of phi:
%
test = norm(( phi'*M*phi - eye(size(phi,2)) ),'fro')/norm(phi'*M*phi,'fro' );
if test > 10^(-4);
  warning = ['Orthonormality of phi is in question.  The norm of phi'*M*phi '...
             'over (phi'*M*phi - I) is ' num2str(test) '.  A significant '...
             'value indicates discrepancies.  This often occurs with two '...
             'rigid body modes since the eigenvector extraction doesn't ensure'...
             'orthogonality of the two rigid eigenvectors.']
end
%
y = []; I = []; test = []; warning = [];
% _____________________________________________________________________
% Section (5)
% Label frequencies as rigid body or bending mode.
% DEFINE: rlab.
% USE: fv, n2.
% .....................................................................
%
if nargout == 3;% Check to see if rlab is requested.
  rlab = [];% Define a null row label.
  nrb = 0;% Initialize the number of rigid body modes.
  if n2 >= 3
    if abs(fv(3))/abs(fv(1)) > 5000; % A reasonable numeric zero.
      rlab = str2mat(rlab, 'Rigid_Body_Mode');
      nrb = 1;
      %
      if abs(fv(3))/abs(fv(2)) > 5000; % A reasonable numeric zero.
        rlab = str2mat(rlab, 'Rigid_Body_Mode');
        nrb = 2;
      end
    end
  else % If fewer than 3 modes, identifying the r-body mode is skipped.
    for I = 1:n2
      rlab = str2mat(rlab, ['Mode_' int2str(I)]);
    end
    nrb = n2;% This prevents the redundant assignment of bending mode
    %        row labels.
  end
  %
  for I = 1:(n2-nrb)
    rlab = str2mat(rlab, ['Bend_Mode_' int2str(I)]);
  end
end
%
rlab(1,:) = [];% Nullify initial empty string.
%
nrb = []; I = [];
% _____________________________________________________________________
% Completed:
% OUTPUT: fv, phi, rlab.
% .....................................................................
%
```

<fem2ss.m>

```
function [A,B,C,D] = fem2ss(M,K,Cd,tlabel)
% fem2ss.
% [A,B,C,D] = fem2ss(M,K,Cd,tlabel)
% o   This m-file generates the first order state-space model from the
%     mass, stiffness, and damping matrices.
% o   Specific variable definitions are:
%   M, K, Cd -> n2xn2 mass and stiffness and damping matrices where n2 = number
%     n2 = number of generalized coordinates.
%   A, B, C, D -> State-Space Matrices as in:
%         dqv = A*qv + B*uv
%         yv  = C*qv + D*uv ... where qv = state vector
%                              uv = input vector
%                              yv = output vector
%   tlabel -> Optional string to enable, and label, matrix printing.
%           Note: labels only valid for beam generalized coordinates.
% o   Created 14 September - 05 October 1995 by Eric Kathe.
%   Benet Labs, Watervliet Arsenal, NY  12189-4050 <ekathe@pica.army.mil>
% ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
% _____________________________________________________________________
% Section (1)
% Define the number of state variables and generalized coordinates.
% DEFINE: n2, nq.
% USE: alpha, beta, M, K.
% .....................................................................
%
n2 = size(M,1);% # of generalized coordinates.
nq = 2*n2;% # of state variables.
%
% _____________________________________________________________________
% Section (2)
```

```
% Compute the first order matrices as in:
% [ dx ] = [    0       I   ] [ x ] + [   0   ]
% [ddx ] = [-inv(M)*K -inv(M)*C ] [dx ] + [ inv(M) ] [ F ]
% ------------------------------------------------------
%  dq  =        A     *     q   +     B  *  u
%
%  &    y = C*q + D*u; Where I'll use the convention [y] = [x].
%
% DEFINE: A, B, C, D.
% USE: Cd, M, K.
% ..........................................................
%%
A = [zeros(n2,n2) eye(n2); -M\K -M\Cd]; % Matrix division preferred to
B = [zeros(n2,n2); inv(M)];             % explicit inversion.(Ref. Guide)
C = [eye(n2) zeros(n2,n2)];
D = zeros(n2,n2);
%
% ____________________________________________________________
% Section (4)
% If tlabel is set, print the state-space matrices.
% USE: C, K, M, n2, nargin, tlabel, author provided naming convention.
% ..........................................................
%
if nargin == 4
 if isstr(tlabel); % Checks to see if tlabel is a string.
  if strcmp(tlabel,'tlabel'); % Checks to see if tlabel is used.
   tlabel = ['Beam']; %       If so, a default is applied.
  else
   tlabel(1) = upper(tlabel(1)); % Imposes upper case on first letter.
  end
 else
  tlabel = ['Beam']; % Default is applied in no valid tlabel given.
 end
 name = [tlabel ' State-Space Matrix '];
 dqlab = [];
 qlab = [];
 ulab = [];
 ylab = [];
 for I = 1:n2/2
  qlab = [qlab 'x' int2str(I) ' '];
  qlab = [qlab 'theta' int2str(I) ' '];
  dqlab = [dqlab 'dx' int2str(I) ' '];
  dqlab = [dqlab 'dtheta' int2str(I) ' '];
  ulab = [ulab 'F' int2str(I) ' '];
  ulab = [ulab 'M' int2str(I) ' '];
 end
 ylab = qlab;
 qlab = [qlab dqlab];
 for I = 1:n2/2
  dqlab = [dqlab 'ddx' int2str(I) ' '];
  dqlab = [dqlab 'ddtheta' int2str(I) ' '];
 end
 %
 printmat(A,[name 'A'],dqlab,qlab)
 printmat(B,[name 'B'],dqlab,ulab)
 printmat(C,[name 'C'],ylab,qlab)
 printmat(D,[name 'D'],ylab,ulab)
end
% ____________________________________________________________
% Completed:
% OUTPUT: A, B, C, D.
% ..........................................................
%
```

<fem_beamel.m>

```
function [Me,Ke]=fem_beamel(lengv,lrho,lEI)
% fem_beamel.m
% [Me,Ke]=fem_beamel(lengv,lrho,lEI)
% o Element Matrices for Euler-Bernoulli Beam (FEM approach)
%   undergoing transverse vibration.
% o This file uses:
%     geom_check.m in section 1.
%     fem_interp.m in section 4.
% o Input/Output Variables:
%   Me -> 4x4 element mass matrix for coordinates
%         (x1,theta1,x2,theta2)
%   Ke -> 4x4 element stiffness matrix for coordinates
%         (x1,theta1,x2,theta2)
%   lengv -> Vector of input element positions, evenly spaced.
%            (commonly in millimeters from 1mm to next node-this node.)
%   lrho -> variable linear density at positions that correspond to
%           lengv.
%   lEI -> variable EI (bending resistance) at positions that correspond
%          to lengv.
% o This file was created using elebeam.m as a guide, which was programmed
%   by Youdan Kim, Dept. of Aerospace Eng., Texas A&M University.
% o The integration technique used here to form the inertial and
%   stiffness matrices of each 'finite' element of the beam is drawn
%   from: Junkins, "Introduction to Dynamics and Control of Flexible
%   Structures," equations 4.101 and 4.102.
% o Created 31 August - 6 October 1995 by Eric Kathe.
%   Benet Labs, Watervliet Arsenal, NY  12189-4050 <ekathe@pica.army.mil>
% ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
% ____________________________________________________________
% Section (1)
% A few checks to be sure input variables are the right size, et ceters.
% This section uses geom_check.m to validate the input geometry vectors.
% USE: lengv,lrho,lEI.
% POSSIBLY ALTER: lengv,lrho,lEI.
% ..........................................................
%
[lengv,lrho,lEI] = geom_check(lengv,lrho,lEI);%
% ____________________________________________________________
% Section (2)
% Extract several values from the data, including the spatial resolution.
% DEFINE: deltal, h, nl.
% USE: lengv.
% ..........................................................
%
deltal = mean(diff(lengv));
h = max(lengv);
nl = length(lengv);
% ____________________________________________________________
% Section (3)
% If the geometry vectors of the element are short, the data is stretched
% to make it longer for higher resolution of the inner-product integral
% approximation.  This zero-order-hold like approach preserves total mass.
% DEFINE: mss.
% POSSIBLY ALTER: deltal, lEI, lengv, lrho.
% USE: deltal, lEI, lengv, lrho, nl, author provided default value.
% ..........................................................
%
mss = 50; % Author defined minimum sample size or spatial resolution.
%
if nl < mss;
  mf = ceil(mss/nl); % integer magnification factor.
  rsnl = nl*mf; % New length is of {(mss+1):(2*mss-1)}.
  rsindex = (1:rsnl)'; % Set the new resample index column vector.
  %
  rsdeltal = deltal/mf; % Redefine the new resample deltal.
  rslengv = rsdeltal*rsindex; % Define new resample lengv.
  %
  rslrhom = lrho(:,ones(1,mf))';% Make a matrix of repeated columns.
  rslrho = rslrhom(:);% Convert matrix to single column vector with each
  %                   value repeated mf times. (Like a zero-order-hold.)
  %
  rslEIm = lEI(:,ones(1,mf))';% As above.
  rslEI = rslEIm(:);
  %
  deltal = rsdeltal; % Note: by the change of deltal, the resample
  lengv = rslengv;   % effect on lrho & lEI is accounted for in the.
  lrho = rslrho;     % integral approximation of section 5.
  lEI = rslEI;
end
%
mf = []; rsdeltal = []; rsindex = []; rslengv = []; rslrho = [];
rslEI = []; rsnl = [];
% ____________________________________________________________
% Section (4)
% Define and numerically evaluate the interpolation functions and
% derivatives at the resolution of the input spatial vector, lengv.
% DEFINE: zphi, zddphi.
% USE lengv.
% ..........................................................
%
[zphi, zddphi] = fem_interp(lengv);
% ____________________________________________________________
% Section (5)
% Numerically evaluate the integrals of Junkins, eq((4.102) using
% an inner product estimation for the integrals.
% DEFINE: Me, Ke.
% USE: deltal, lEI, lrho, zphi, zddphi.
% ..........................................................
%
Me = zeros(4,4); Ke = zeros(4,4);% Initialize the matrices.
for I = 1:4
 for j = 1:i
   Me(i,j) = sum(lrho.*zphi(:,I).*zphi(:,j))*deltal;
   Ke(i,j) = sum(lEI.*zddphi(:,I).*zddphi(:,j))*deltal;
 end
end
%
```

```
108 for I = 1:3
109   for j = (i+1):4
110     Me(i,j)=Me(j,i);
111     Ke(i,j)=Ke(j,i);
112   end
113 end
114 % _____
115 % Completed:
116 % OUTPUT: Me, Ke.
117 % .....
```

<fem_force.m>

```
1   function [F] = fem_force(spatial,lforce,lmoment,ncord,tlabel)
2   % fem_force.m
3   % [F] = fem_force(spatial,lforce,lmoment,ncord,tlabel);
4   % o  This operator computes the force vector, F,
5   %    for an Euler-Bernoulli beam using the FEM with standard
6   %    reaction force techniques.
7   % o  The input geometry vector spatial is generally entered and computed
8   %    using geomf_*.m.
9   % o  The node index vector, ncord is generally entered and computed using
10  %    fem_mesh.m.
11  % o  This file uses:
12  %       geom_check.m in section 1
13  %       fem_node_check.m in section 2
14  %         geom_check.m
15  % o  Specific variable definitions are:
16  %  spatial -> Axial position vector, currently limited to equally spaced
17  %    position vectors with position data of every dx*n point such that
18  %    n = 1:length(spatial) and length(spatial)*dx equals max(spatial).
19  %  lforce -> Beam transverse force vector such that each index value
20  %    corresponds to a lateral force imposed at the beam position of the
21  %    respective spatial value at the same index.
22  %  lmoment -> similar to lforce except for moment loading in lieu of
23  %    transverse force loading.
24  %  ncord -> The vector of indices such that spatial(ncord) is the position
25  %    of each pair of (nel+1) generalized coordinates.
26  %  tlabel -> Enables printing of the force vector.
27  %    If valid, the string of tlabel is incorporated into the listing to
28  %    describe the beam.
29  %  F -> Out-put force vector of the second-order self-adjoint
30  %    system of size 2*(nel+1) by 1.  (The generalized coordinate
31  %    vector in this case consists of [x1 theta1 x2 theta2 ..theta(nel+1)]'.
32  % o  Created 05 October 1995 by Eric Kathe.
33  %  Benet Labs, Watervliet Arsenal, NY  12189-4050 <ekathe@pica.army.mil>
34  % ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
35  % _____
36  % Section (1)
37  % A few checks to be sure input variables are the right size, et cetera.
38  % This section uses geom_check.m to validate the input geometry vector.
39  % Also define the number of elements of spatial, ns.  The ncord will be
40  % checked later, in section 2.
41  % DEFINE: ns.
42  % USE: lfoce, lmoment, spatial.
43  % POSSIBLY ALTER: lfoce, lmoment, spatial.
44  % .....
45  %
46  [spatial] = geom_check(spatial);
47  ns = length(spatial); % number of elements of spatial.
48  %
49  lforce = lforce(:);% Impose column structure on vector.
50  if length(lforce) ~= ns
51    zvector = zeros(ns,1);
52    zvector(size(lforce)) = lforce;% If lforce larger zvector is too big,
53    lforce = zvector(1:ns);%       then it is truncated, else zero pad.
54    warning = ['Lforce must be same length as spatial.  It has been'...
55          'padded or truncated to size.  Errors likely.']
56  end
57  %
58  lmoment = lmoment(:);% Impose column structure on vector.
59  if length(lmoment) ~= ns
60    zvector = zeros(ns,1);
61    zvector(size(lmoment)) = lmoment;% If lmoment larger zvector is too big,
62    lmoment = zvector(1:ns);%       then it is truncated, else zero pad.
63    warning = ['Lmoment must be same length as spatial.  It has been'...
64          'padded or truncated to size.  Errors likely.']
65  end
66  %
67  warning = []; zvector = [];
68  % _____
69  % Section (2)
70  % Validate & sort node vector, ncord.  Also determine the number
71  % of nodes, nn, and number of elements, nel.
72  % This section uses fem_node_check.m to validate the node vector.
73  % DEFINE: nn, nel.
74  % USE: ncord, ns.
75  % POSSIBLY ALTER: ncord.
76  % .....
77  %
78  [ncord,spatial] = fem_node_check(ncord,spatial);
79  nn = length(ncord); % number of nodes of ncord.
80  %
81  nel = nn - 1;
82  % _____
83  % Section (3)
84  % Determine the equivalent node force equation for lateral loading.
85  %
86  %                    f
87  %              a     ^     b
88  %  _ -> |<------- x -------> | <---------- h - x ---------->|<- _
89  %  /  ___________________|_____________________________________   \
90  % | ML  |XXXXXXXXXXXXXXXXXXXX BEAM ELEMENT
XXXXXXXXXXXXXXXXXXXX | MR  |
91  % \_   ^-------------------------------------------------^   _/
92  %    -__ | <---------------------- h ----------------------> | _-
93  %      |                 L                 |
94  %      FL                                  FR
95  %
96  % USE: Author provided data as referenced.
97  % Define: mlf, flf, mrf, frf.
98  % .....
99  %
100 % Formulae from: Sennett,"Matrix Analysis of Structures," Prentice
101 %            Prentice-Hall, 1994, page 69. (Alt. reference below.)
102 %
103 flf = '(f).*((h-x).^2).*(2*x+h)/(h^3)';% Dot notation allows for the use
104 mlf = '(f).*(x).*((h-x).^2)/(h^2)';%    of the eval command in section 5.
105 frf = '(f).*(x.^2).*(3*h-2*x)/(h^3)';
106 mrf = '(f).*(x.^2).*(h-x)/(h^2)';
107 % _____
108 % Section (4)
109 % Determine the equivalent node force equation for moment loading.
110 %
111 %              a       |       b
112 %  _ -> |<------- x -------> _ <---------- h - x ---------->|<- _
113 %  /  ___________________/  \ _____________________________   \
114 % | ML  |XXXXXXXXXXXXXXXXX | m  XXX BEAM ELEMENT XXXXXXXXXXXX |
MR  |
115 % \_   ^-----------------\_ -> --------------------------^   /
116 %    -__ | <---------------------- h ----------------------> | _-
117 %      |                 L                 |
118 %      FL                                  FR
119 %
120 % USE: Author provided data as referenced.
121 % Define: mlm, flm, mrm, frm.
122 % .....
123 %
124 % Formulae from: Buchanan,"Theory and Problems of Finite Element
125 %            Analysis," McGraw-Hill, Schaum's, 1995, page 111.
126 %            (Alt reference above.)
127 %
128 flm = '-6*(m).*(x).*(h-x)/(h^3)';
129 mlm = '(m).*(h-x).*((h-x)-2*x)/(h^2)';
130 frm = '6*(m).*(x).*(h-x)/(h^3)';
131 mrm = '(m).*(x).*(x-2*(h-x))/(h^2)';
132 %
133 % _____
134 % Section (5)
135 % Evaluate the force vector for each finite element and combine
136 % into a complete system force vector.
137 % DEFINE: Fv.
138 % USE: lforce, lmoment, ncord, nel, nn, spatial, mlm, flm, mrm, frm
139 %    mlf, flf, mrf, frf.
140 % .....
141 %
142 % (A) Define the number of generalized coordinates and initialize the
143 %   force vector, F:
144 %
145 n2 = 2*nn;
146 F = zeros(n2,1);
147 %
148 % (B) For the first element of the beam:
149 %
150 x = spatial(ncord(1):ncord(2)); % Length vector of finite element.
151 f = lforce(ncord(1):ncord(2)); % Force vector of finite element.
152 m = lmoment(ncord(1):ncord(2)); % Moment vector of finite element.
153 h = max(x);% Length of element.
154 %
155 FVe = [sum(eval(flf) + eval(flm));% The first elements contribution to F.
156        sum(eval(mlf) + eval(mlm));
157        sum(eval(frf) + eval(frm));
158        sum(eval(mrf) + eval(mrm))];
```

```
159 %
160 F(1:4) = FVe;% Adds in the contribution of the first element to F.
161 %
162 %
163 % © For remaining elements:
164 %   (The looping sum algebraically couples the elemental node forces
165 %    by overlapping them when defining the system force vector.)
166 %
167 for l=2:nel;
168  %
169  x = spatial((ncord(l)+1):ncord(l+1)) - spatial(ncord(l));
170  f = lforce((ncord(l)+1):ncord(l+1)); % Force vector of finite element.
171  m = lmoment((ncord(l)+1):ncord(l+1)); % Moment vector of finite element.
172  h = max(x);% Length of element.
173  %
174  FVe = [sum(eval(flf) + eval(flm));% Later elements contribution to F.
175       sum(eval(mlf) + eval(mlm));
176       sum(eval(frf) + eval(frm));
177       sum(eval(mrf) + eval(mrm))];
178  %
179  ind1=2*l-1; ind2=ind1+3;
180  F(ind1:ind2) = F(ind1:ind2) + FVe;
181 end
182 %
183 l = []; ind1 = []; ind2 = []; x = []; f = []; h = []; m = [];
184 n2 = []; FVe = [];
185 % ____________________________________________________________
186 % Section (6)
187 % If tlabel is set, print the nodal force vector.
188 % USE: F, nn, nargin, tlabel, author provided naming convention.
189 % POSSIBLY MODIFY: tlabel.
190 % ..........................................................
191 %
192 if nargin == 5
193  if isstr(tlabel); % Checks to see if tlabel is a string.
194   if strcmp(tlabel,'tlabel'); % Checks to see if tlabel is used.
195    tlabel = ['Beam']; %      If so, a default is applied.
196   else
197    tlabel(1) = upper(tlabel(1)); % Imposes upper case on first letter.
198   end
199  else
200   tlabel = ['Beam']; % Default is applied if no valid tlabel given.
201  end
202  name = [tlabel ' Nodal Force Vector'];
203  ulab = {};
204  for I = 1:nn
205   ulab = [ulab 'F' int2str(I) ' '];
206   ulab = [ulab 'M' int2str(I) ' '];
207  end
208  %
209  printmat(F,name,ulab,[])
210 end
211 %
212 ulab = [];
213 % ____________________________________________________________
214 % Completed:
215 % OUTPUT: F.
216 % ..........................................................
217 %
```

<fem_form.m>

```
1  function [M,K] = fem_form(spatial,lden,lEI,lnbden,ncord,tlabel)
2  % fem_form.m
3  % [M,K] = fem_form(spatial,lden,lEI,lnbden,ncord,tlabel)
4  % o  This m-file generates the mass and stiffness matrices for arbitrary
5  %    beams as described in Junkins, "Introduction to Dynamics
6  %    and Control of flexible Structures," pp. 200
7  % o  This operator computes the mass and stiffness matrices (M,K)
8  %    for an Euler-Bernoulli beam using the FEM with standard
9  %    (cubic spline) interpolation functions.  M and K are the
10 %    symmetric, positive definite (or semi-definite) coefficient matrices
11 %    in the linear system: M ddxv + C dxv + K xv = fv
12 % o  The input geometry is generally entered and computed using
13 %    *_geomf.m.
14 % o  The node index vector, ncord is generally entered and computed using
15 %    fem_mesh.m.
16 % o  This file uses:
17 %       geom_check.m in section 1
18 %       fem_node_check.m in section 2
19 %        geom_check.m
20 %       fem_beamel.m in section 4
21 %        geom_check.m
22 %        fem_interp.m
23 %         geom_check.m
24 % o  Non beam masses are also included.
25 % o  Specific variable definitions are:
26 %   spatial -> Axial position vector, currently limited to equally spaced
27 %     position vectors with position data of every dx*n point such that
28 %     n = 1:length(spatial) and length(spatial)*dx equals max(spatial).
29 %   lden -> Beam linear density vector such that each index value
30 %     corresponds to the beam position of the respective spatial value
31 %     at the same index.
32 %   lEI -> similar to lden except for linear EI cross-section properties.
33 %   lnbden -> similar to lden, except this records the inertia of non-beam
34 %     masses that are attached to the beam.
35 %   ncord -> The vector of indices such that spatial(ncord) is the position
36 %     of each 2*(nel+1) generalized coordinates. (Also thought of as nodes.)
37 %   tlabel -> Enables plotting of the beam mass and stiffness matrices.
38 %     If valid, the string of tlabel is incorporated into the plot to
39 %     describe the beam.
40 %   M,K -> Out-put mass and stiffness matrices of the second-order self-adjoint
41 %     system commonly of size 2*(nel+1) square.  (The generalized coordinate
42 %     vector in this case consists of [x1 theta1 x2 theta2 ..theta(nel+1)]'.
43 % o  This file was created using beamfem.m as a guide, which was programmed
44 %    by Youdan Kim, Dept. of Aerospace Eng., Texas A&M University.
45 % o  Created 13 September - 24 October 1995 by Eric Kathe.
46 %   Benet Labs, Watervliet Arsenal, NY  12189-4050 <ekathe@pica.army.mil>
47 % ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
48 % ____________________________________________________________
49 % Section (1)
50 % A few checks to be sure input variables are the right size, et ceters.
51 % This section uses geom_check.m to validate the input geometry vectors.
52 % Also define the number of elements of spatial, ns.  The ncord will be
53 % checked later, in section 2.
54 % DEFINE: ns.
55 % USE: lden, lEI, lnbden, spatial.
56 % POSSIBLY ALTER: lden, lEI, lnbden, spatial.
57 % ..........................................................
58 %
59 [spatial,lden,lEI,lnbden] = geom_check(spatial,lden,lEI,lnbden);
60 ns = length(spatial); % number of elements of spatial.
61 % ____________________________________________________________
62 % Section (2)
63 % Validate & sort node vector, ncord.  Also determine the number
64 % of nodes, nn, and number of elements, nel.
65 % This section uses fem_node_check.m to validate the node vector.
66 % DEFINE: nn, nel.
67 % USE: ncord, ns.
68 % POSSIBLY ALTER: ncord.
69 % ..........................................................
70 %
71 [ncord,spatial] = fem_node_check(ncord,spatial);
72 nn = length(ncord); % number of nodes of ncord.
73 %
74 nel = nn - 1;
75 % ____________________________________________________________
76 % Section (3)
77 % Combine beam and non-beam linear densities:
78 % DEFINE: ldent.
79 % USE: lden, lnbden.
80 % ..........................................................
81 %
82 ldent = lden + lnbden; % Net linear density including beam and non-beam.
83 %
84 % ____________________________________________________________
85 % Section (4)
86 % Compute the mass and stiffness matrices for each finite element using
87 % elebeamint.m, and combine into complete system M & K matrices.
88 % DEFINE: K, M.
89 % USE: ldent, lEI, ncord, nel, nn, spatial
90 % ..........................................................
91 %  For the first element of the beam
92 %
93 lengvseg = spatial(ncord(1):ncord(2)); % length of finite element.
94 lrhoseg = ldent(ncord(1):ncord(2)); % linear density of finite element.
95 lEIseg = lEI(ncord(1):ncord(2)); % EI distribution over finite element.
96 %
97 [Me,Ke] = fem_beamel(lengvseg, lrhoseg, lEIseg); % element M&K.
98 %
99 ngce = size(Me,1); % The number of elemental generalized coordinates.
100 %          (Commonly four corresponding to: [xL thetaL xR thetaR].)
101 %
102 n2=(ngce/2)*(nn);% n2 is the total number of generalized coordinates.
103 %
104 %  Initialize the Matrices
105 %
106 M = zeros(n2,n2);
107 K = zeros(n2,n2);
108 %
109 M(1:ngce,1:ngce) = Me;
110 K(1:ngce,1:ngce) = Ke; % Add in element M&K into total M&K.
111 %
```

```
112 %  For remaining elements:
113 %  (The keeping sum algebraically couples the elemental matrices
114 %   by overlapping them when defining the system matrices.)
115 %
116 for I=2:nel;
117  %
118  lengvseg = spatial((ncord(I)+1):ncord(I+1)) - spatial(ncord(I));
119  lrhoseg = ident((ncord(I)+1):ncord(I+1));
120  lEIseg = lEI((ncord(I)+1):ncord(I+1));
121  %
122  [Me,Ke] = fem_beamel(lengvseg, lrhoseg, lEIseg);
123  %
124  ind1=2*I-1; ind2=ind1+3;
125  ind1=(ngce/2)*(I-1) + 1;
126  ind2=ind1 + ngce - 1;
127  M(ind1:ind2,ind1:ind2)=M(ind1:ind2,ind1:ind2)+Me;
128  K(ind1:ind2,ind1:ind2)=K(ind1:ind2,ind1:ind2)+Ke;
129 end
130 %
131 I = []; ind1 = []; ind2 = []; Ke = []; Me = []; n2 = []; ngce = [];
132 % _____
133 % Section (5)
134 % If tlabel is set, plot the M & K matrices for visual inspection.
135 % the current figure window.
136 % USE: ncord, nmperk, spatial, sv.
137 % ..........
138 %
139 if nargin == 6
140  clg, colormap(flipud(hot))
141  if isstr(tlabel); % Checks to see if tlabel is a string.
142   if strcmp(tlabel,'flag'); % Checks to see if tlabel is an old
143    tlabel = ['Beam']; %     convention.  If so, a default is applied.
144   elseif strcmp(tlabel,'tlabel'); % Checks to see if tlabel is used.
145    tlabel = ['Beam']; %              If so, a default is applied.
146   else
147    tlabel(1) = upper(tlabel(1)); % Imposes upper case on first letter.
148   end
149  else
150   tlabel = ['Beam']; % Default is applied in no valid tlabel given.
151  end
152  subplot(321), image(64*M/max(max(M))); % Normalize M to 0 to 64.
153  title(['Image of ' tlabel ' Mass Matrix'])
154  subplot(11,2,11), image(1:64)
155  title('Color Scale')
156  alabel = ['Zero';'High'];
157  set(gca,'XTick',[1,64])
158  set(gca,'XTickLabels',alabel)
159  set(gca,'YTick',[])
160  subplot(325), image(64*K/max(max(K))) % Normalize K to 0 to 64.
161  title(['Image of ' tlabel ' Stiffness Matrix'])
162  subplot(222), spy(M)
163  title('On/Off Image of Nonzero Mass Matrix Elements')
164  subplot(224), spy(K)
165  title('On/Off Image of Nonzero Stiffness Matrix Elements')
166  orient('tall')
167 end
168 % _____
169 % Completed:
170 % OUTPUT: M, K.
171 % ..........
172 %
```

<fem_interp.m>

```
1  function [zphi,zddphi] = fem_interp(lengv)
2  % fem_interp.m
3  % [zphi,zddphi] = fem_interp(lengv)
4  % o This function numerically computes the interpolation functions
5  %   for finite element formulation of a Euler-Bernoulli Beam
6  %   undergoing transverse vibration.
7  % o This file uses:
8  %    geom_check.m in section 1.
9  % o This version does not use the symbolic toolbox.
10 % o Input/Output Variables:
11 %   lengv -> Vector of input element positions, evenly spaced.
12 %           (Commonly in millimeters from 1mm to element length.)
13 %   zphi, zddphi -> The numerical evaluation of the interpolation
14 %                   functions and their derivatives in a form compatible
15 %                   with lengv.
16 % o The method used here is drawn from: Junkins,
17 %   "Introduction to Dynamics and Control of Flexible
18 %    Structures," equation 4.101.
19 % o Created 31 August - 06 October 1995 by Eric Kathe.
20 %   Benet Labs, Watervliet Arsenal, NY 12189-4050 <ekathe@pica.army.mil>
21 %   ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
22 %  _____
23 % Section (1)
24 % Check to be sure lengv is well posed.
25 % USE: lengv.
26 % POSSIBLY ALTER: lengv.
27 % ..........
28 %
29 lengv = geom_check(lengv);
30 % _____
31 % Section (2)
32 % Extract two values from lengv.
33 % DEFINE: h, nl.
34 % USE: lengv.
35 % ..........
36 %
37 h = max(lengv);
38 nl = length(lengv);
39 % _____
40 % Section (3)
41 % Define the cubic spline interpolation functions and their 2nd spatial
42 % derivatives: (Note: the .^'s provide for element by element computation
43 %              for x (a vector) in the eval command of section 4.)
44 % (Note: the symbolic toolbox is an excellent means to arrive at the
45 %  interpolation function derivatives.  This means has been removed to
46 %  facilitate more general use.)
47 % DEFINE: phi1, phi2, phi3, phi4, ddphi1, ddphi2, ddphi3, ddphi4.
48 % USE: author provided symbolic spline functions.
49 % ..........
50 %
51 phi1 = '1 - 3*x.^2/(h^2) + 2*x.^3/(h^3)';
52 phi2 = 'x - 2*h*x.^2/(h^2) + h*x.^3/(h^3)';
53 phi3 = '3*x.^2/(h^2) - 2*x.^3/(h^3)';
54 phi4 = '-h*x.^2/(h^2) + h*x.^3/(h^3)';
55 %
56 ddphi1 = '-6/h^2+12*x/h^3';
57 ddphi2 = '-4/h+6*x/h^2';
58 ddphi3 = '6/h^2-12*x/h^3';
59 ddphi4 = '-2/h+6*x/h^2';
60 % _____
61 % Section (4)
62 % Numerically evaluate the interpolation functions and derivatives at the
63 % resolution of the input spatial vector, lengv.
64 % DEFINE: zphi, zddphi.
65 % USE: lengv, nl, phi1, phi2, phi3, phi4, ddphi1, ddphi2, ddphi3, ddphi4.
66 % INDIRECTLY USE: h (ie, it's embedded in the eval commands.)
67 % NULLIFY: phi1, phi2, phi3, phi4, ddphi1, ddphi2, ddphi3, ddphi4.
68 % ..........
69 %
70 % Initialize matrices to hold numeric evaluations of interpolation functions:
71 zphi = zeros(nl,4);
72 zddphi = zeros(nl,4);
73 %
74 x = lengv;
75 %
76 zphi(:,1) = eval(phi1);
77 zphi(:,2) = eval(phi2);
78 zphi(:,3) = eval(phi3);
79 zphi(:,4) = eval(phi4);
80 %
81 zddphi(:,1) = eval(ddphi1);
82 zddphi(:,2) = eval(ddphi2);
83 zddphi(:,3) = eval(ddphi3);
84 zddphi(:,4) = eval(ddphi4);
85 %
86 phi1 = []; phi2 = []; phi3 = []; phi4 = [];
87 ddphi1 = []; ddphi2 = []; ddphi3 = []; ddphi4 = [];
88 x = [];
89 % _____
90 % Completed:
91 % OUTPUT: zphi, zddphi.
92 % ..........
93 %
```

<fem_lump.m>

```
1  function [M,K,Cd] = fem_lump(M,K,Mextl,Mextr,alpha,beta,constraintm,mtdm)
2  % fem_lump.
3  % [M,K,Cd] = fem_lump(M,K,alpha,beta,constraintm,mtdm);
4  % o  This m-file augments the finite element formulation of the
5  %    Mass and Stiffness matrices with external lumped parameter
6  %    elements and assembles the proportional damping matrix.
7  % o  This file uses:
8  %     fem_lumpm_check.m in Section 1.
9  % o  Specific variable definitions are:
10 %  M, K (Input) -> n2xn2 mass and stiffness matrices where n2 = number
11 %       of generalized coordinates.  These include only the beam
12 %       dynamics with no constraints.  (If mass tuned damper elements are
13 %       included n2 will increase by the number of rows of mtdm.)
14 %  Mextl, Mextr -> 2x2 sub matrices of left and right extreme rigid body
```

```
%        inertia. (Warn & zeros will be substituted if not 2x2.)  These
%        are generally entered and computed using geomf_*.m.
%   alpha, beta -> scaler proportionality constants for the Rayleigh
%        damping matrix, Cd = alpha*M + beta*K. (Shames & Dyn, pp. 646.)
%   constraintm -> Translational or rotational lumped parameter constraint
%        spring and damping matrix.  The matrix is assembled one row for
%        each node effected.  Column1 -> generalized coordinate number,
%                             NOTE: Must be in ascending order.
%                        Column2 -> lumped spring constant,
%                        Column3 -> lumped damping coefficient.
%   mtdm -> Mass tuned damper matrix. The structure is the same as
%        constraintm with an additional column4 -> lumped mtd mass.
%   M, K, Cd (Output) -> Mass, stiffness, and Rayleigh damping matrices of
%        generalized coordinates that include the constraint and
%        externally coupled dynamics.  (If mass tuned damper elements
%        are included n2 will increase by the number of rows of mtdm.)
% o  Created 29 September - 31 October 1995 by Eric Kathe.
%  Benet Labs, Watervliet Arsenal, NY  12189-4050 <ekathe@pica.army.mil>
% ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
% _____________________________________________________________________________
% Section (1)
% Check the damping coefficients to ensure that they are not negative.
% Also check Mextl, Mextr, constraintm, mtdm.  Warnings indicate reasons
% for if checks.
% USE: alpha, beta, constraintm, mtdm, Mextl, Mextr.
% Define: n2, ncnst, nmtd.
% POSSIBLY ALTER: alpha, beta, constraintm, mtdm, Mextl, Mextr.
% .............................................................................
%
% (A) Check alpha & beta.
%
if alpha < 0
  alpha = 0;
  warning = 'Negative alpha not valid.  Alpha set to zero.'
end
if beta < 0
  beta = 0;
  warning = 'Negative beta not valid.  Beta set to zero.'
end
%
% (B) Check sizes of Mextl & Mextr.
%
[m,n] = size(Mextl);
if m ~= 2
  Mextl = zeros(2,2);
  warning = 'Mextl must be 2x2.  It has been set to 2x2 zeros.'
elseif n ~= 2
  Mextl = zeros(2,2);
  warning = 'Mextl must be 2x2.  It has been set to 2x2 zeros.'
end
[m,n] = size(Mextr);
if m ~= 2
  Mextr = zeros(2,2);
  warning = 'Mextr must be 2x2.  It has been set to zeros.'
elseif n ~= 2
  Mextr = zeros(2,2);
  warning = 'Mextr must be 2x2.  It has been set to zeros.'
end
%
% © First check of constraintm for proper input form.
%
constraintm = fem_lumpm_check(constraintm);
%
% (D) Second check of constraintm for comparison against M.
%    Also define the number of generalized coordinates, n2,
%    and the number of external constraints, ncnst.
%
n2 = size(M,1);% The number of generalized coordinates.
if size(constraintm,1) > n2
  constraintm = constraintm(1:n2,:);
  warning = ['Too many constraints, constraintm truncated to '...
          'the number of generalized coordinates (' ...
          int2str(n2) ')']
end
ncnst = size(constraintm,1);% The number of external constraints.
%
% (E) Check the mass tuned damper & compare against M.
%    Also define the number of mtd's. Note: This input is
%    optional.  Its presence is checked via nargin (the number
%    of input variables).
%
nmtd = 0;% Initialize the number of mass tuned dampers to zero.
if nargin == 8
  mtdm = fem_lumpm_check(mtdm);
  if size(mtdm,1) > n2
    mtdm = mtdm(1:n2,:);
    warning = ['Too many mass tuned dampers, mtdm truncated to '...
          'the number of generalized coordinates (' ...
          int2str(n2) ')']
  end
  nmtd = size(mtdm,1);
end
%
m = []; n = []; warning = [];
% _____________________________________________________________________________
% Section (2)
% Compute the damping matrix and add Mextl&r to M.
% DEFINE: Cd.
% USE: alpha, beta, M, K, n2, Mextl, Mextr.
% .............................................................................
%
M(1:2,1:2) = M(1:2,1:2) + Mextl;
M((n2-1):n2,(n2-1):n2) = M((n2-1):n2,(n2-1):n2) + Mextr;
%
Cd = alpha*M + beta*K; % Proportional damping matrix.
% _____________________________________________________________________________
% Section (3)
% Add in the external constraint stiffness and damping.
% POSSIBLY MODIFY: Cd, K, M.
% USE: Cd, constraintm, M, K, ncnst.
% .............................................................................
%
for I = 1:ncnst
  gcind = constraintm(i,1);
  K(gcind,gcind) = K(gcind,gcind) + constraintm(i,2);
  Cd(gcind,gcind) = Cd(gcind,gcind) + constraintm(i,3);
end
%
gcind = []; I = [];
% _____________________________________________________________________________
% Section (4)
% Add in the external mass tuned dampers.
% POSSIBLY MODIFY: Cd, K, M.
% USE: Cd, M, mtdm, K, nmtd.
% .............................................................................
%
if nargin == 8
  M = [M zeros(n2,nmtd);zeros(nmtd,(n2+nmtd))];
  K = [K zeros(n2,nmtd);zeros(nmtd,(n2+nmtd))];
  Cd = [Cd zeros(n2,nmtd);zeros(nmtd,(n2+nmtd))];
  for I = 1:nmtd
    gcind = mtdm(i,1);% This is the generalized coordinate ,gc, that
    %              the mtd will be coupled to.
    mtdind = n2 + I;%  Mtdind is the new gc of the mtd itself.
    %
    M(mtdind,mtdind) = mtdm(i,4);% Add in new inertia.
    %
    K(mtdind,gcind) = - mtdm(i,2);% Add in new spring effect in four
    K(gcind,mtdind) = - mtdm(i,2);% locations.
    K(gcind,gcind) = K(gcind,gcind) + mtdm(i,2);
    K(mtdind,mtdind) = mtdm(i,2);
    %
    Cd(mtdind,gcind) = - mtdm(i,3);% Add in new damping effect in four
    Cd(gcind,mtdind) = - mtdm(i,3);% locations.
    Cd(gcind,gcind) = Cd(gcind,gcind) + mtdm(i,3);
    Cd(mtdind,mtdind) = mtdm(i,3);
  end
end
%
gcind = []; I = []; mtdind = [];
% _____________________________________________________________________________
% Completed:
% OUTPUT: Cd, K, M.
% .............................................................................
%
```

<fem_lumpm_check.m>

```
function [lumpm] = fem_lumpm_check(lumpm)
% fem_lumpm_check.m
% [lumpm] = fem_lumpm_check(lumpm)
% o  This m-file checks the validity of the lumped parameter matrices
% o  Specific input/output variable definitions are:
%   lumpm -> Translational or rotational lumped parameter matrix.
%        The matrix is assembled one row for each generalized
%        coordinate effected.  Typically:
%                  Column1 -> generalized coordinate number,
%                   Column2 -> lumped spring constant,
%                   Column3 -> lumped damping coefficient,
% o  Created 29 September - 06 October 1995 by Eric Kathe.
%  Benet Labs, Watervliet Arsenal, NY  12189-4050 <ekathe@pica.army.mil>
% ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
% _____________________________________________________________________________
```

```
% Section (1)
% Check first column to be sure that the generalized coordinates are
% referenced by positive integers.
% USE: lumpm.
% POSSIBLY ALTER: lumpm.
% .....
%
if sum(lumpm(:,1) - floor(lumpm(:,1))) > 0
  lumpm(:,1) = floor(lumpm(:,1) + ½);
  warning = ['Column one of lumpm, indicates the generalized coordinate '...
        'or spatial index number and must be integer.  It has been '...
        'reset to: ' int2str(lumpm(:,1))]
end
if min(lumpm(:,1)) < 1;
  lumpm(:,1) = lumpm(:,1) - min(lumpm(:,1)) + 1;% Sets first index to 1.
  warning = ['Column one of lumpm, indicates the generalized coordinate '...
        'number and must be positive.  It has been reset to: '...
        int2str(lumpm(:,1))]
end
%
warning = [];
% _____
% Section (2)
% Insure the rows of lumpm are sorted in order of ascending coordinate
% or spatial index numbers.  Warn if they're shuffled.
% USE: lumpm.
% POSSIBLY ALTER: lumpm.
% .....
%
[y,ind] = sort(lumpm(:,1));
if sum(abs(y - lumpm(:,1))) ~= 0
  warning = 'Lumpm was sorted in order of ascending coordinates.'
end
lumpm = lumpm(ind,:);
%
ind = []; y = [];
% _____
% Section (3)
% Check remaining columns to be sure that the lumped parameters are positive.
% USE: lumpm.
% POSSIBLY ALTER: lumpm.
% .....
%
for I = 2:size(lumpm,2)
  if min(lumpm(:,I)) < 0
    lumpm(:,I) = abs(lumpm(:,I));
    warning = ['Column ' int2str(I) 'of lumpm, indicates a negative '...
        'lumped parameter that must be positive.  It has been '...
        'reset to it"s absolute values: ' num2str(lumpm(:,I))]
  end
end
I = []; warning = [];
% _____
% Completed:
% OUTPUT: lumpm
% .....
%
```

<fem_mesh.m>

```
function [ncord] = fem_mesh(spatial,lden,lEI,lnbden,snlv,nel,tlabel)
% fem_mesh.m
% [ncord] = fem_mesh(spatial,lden,lEI,lnbden,snlv,nel,flag)
% o  This m-file generates the node point coordinate vector for later
%    finite element formulation.
% o  This file uses:
%      geom_check.m
% o  The input geometry is generally entered and computed using
%    geomf_*.m.
% o  Non beam masses (such as the breech) are also included.
% o  Specific input/output variable definitions are:
%  spatial -> Axial position vector, currently limited to equally spaced
%    position vectors with position data of every dx*n point such that
%    n = 1:length(spatial) and length(spatial)*dx equals max(spatial).
%  lden -> Beam linear density vector such that each index value
%    corresponds to the beam position of the respective spatial value
%    at the same index.
%  lEI -> similar to lden except for linear EI cross-section properties.
%  lnbden -> similar to lden, except this records the inertia of non-beam
%    masses that are attached to the beam.
%  snlv -> Imposed node location vector in units of spatial.  Typically
%    consists of two support locations.
%  nel -> Desired number of finite elements, with the condition that s(I)
%    are imposed node points.  The rest of the barrel is broken up in a
%    manner that attempts to evenly space the elements in a cross-sectional
%    property sense.  Minimum value of nel is imposed as length(snlv)+1.
%  tlabel -> Enables plotting of the beam meshing metric, imposed nodes,
%    and the set of all nodes spatial(ncord).  If valid, the string of
%    tlabel is incorporated into the plot to describe the beam.
%  ncord -> The vector of indices such that spatial(ncord) is the position
%    of each pair of (nel+1) generalized coordinates. (Also known as nodes.)
% o  Created 13 September 1995 - 15 January 1996 by Eric Kathe.
%  Benet Labs, Watervliet Arsenal, NY  12189-4050 <ekathe@pica.army.mil>
% ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
%
% _____
% Section (1)
% A few checks to be sure input variables are the right size, et cetera.
% This section uses geom_check.m to validate the input geometry vectors.
% Also define the number of indices of spatial, ns.  The snlv will be
% checked later, in section 2.
% DEFINE: ns.
% USE: lden, lEI, lnbden, nel, spatial, author provided default values.
% POSSIBLY ALTER: lden, lEI, lnbden, nel, spatial.
% .....
%
ns = length(spatial); % number of indices of spatial.
%
[spatial,lden,lEI,lnbden] = geom_check(spatial,lden,lEI,lnbden);
%
neldefault = 14;% Author provided default value to be used in invalid
%              value specified.
%
nel = nel(:);
if length(nel) > 1
  nel = neldefault;
  warning = ['Number of elements, nel, must be a scaler.  '...
        'It has been reset to ' int2str(nel) '.']
end
if nel < 1
  nel = neldefault;
  warning = ['nel must not be negative or zero.  '...
        'It has been reset to ' int2str(nel) '.']
end
if (nel - ceil(nel)) < 0
  nel = ceil(nel-1/2);
  warning = ['nel must be an integer.  '...
        'It has been reset to ' int2str(nel) '.']
end
if nel > (ns - 1)
  nel = (ns - 1);
  warning = ['nel too large, cannot allocate more elements' ...
        'than spatial points.  It is now = ' int2str(nel) '.']
end
%
neldefault = []; warning = []; x = []; y = [];
% _____
% Section (2)
% Validate & Sort imposed node vector.  Also determine the number
% of valid imposed nodes, nin, and number of super segments,
% nsel, between validated imposed nodes, and ends of beam.
% DEFINE: nin, nsel, sv.
% USE: ns, snlv, spatial.
% .....
%
snlv = snlv(:);
snlv = sort(snlv);
nin = length(snlv);% # of imposed nodes. (May be modified later.)
%
if snlv(1) <= mean(spatial(1:2))
  snlv(1) = 0;% Zero values to eliminated later this section.
  warning = 'Negative, zero, or beginning of beam imposed node ignored.'
elseif snlv(1) >= mean(spatial((ns-1):ns))
  snlv(1) = 0;% Zero values to eliminated later this section.
  warning = 'Imposed node beyond or at end of beam ignored.'
end
for I = 2:nin
  if snlv(I) <= mean(spatial(1:2))
    snlv(I) = 0;% Zero values to eliminated later this section.
    warning = 'Negative, zero, or beginning of beam imposed node ignored.'
  elseif snlv(I) >= mean(spatial((ns-1):ns))
    snlv(I) = 0;% Zero values to eliminated later this section.
    warning = 'Imposed node beyond or at end of beam ignored.'
  elseif snlv(I) == snlv((I-1))
    snlv(I) = 0;% Zero values to eliminated later this section.
    warning = 'Repeated imposed node ignored.'
  end
end
%
snlv = sort(snlv(find(snlv)));% Exclude zero elements of snlv & resort.
%
nin = length(snlv); % Final # of valid imposed nodes.
nsel = nin + 1; % # of super segments imposed between valid nodes.
```

```
%
sv = zeros(nin,1); % Initialize imposed node index vector. (Will change.)
for I = 1:nin
  [y, sv(I)] = min(abs(spatial - snlv(I))); % Identify closest index to node.
end
sv = [1; sv; ns]; % Final node vector includes free ends of beam as nodes.
%
I = []; warning = []; y = [];
% _____________________________________________________________________
% Section (3)
% Combine beam and non-beam linear densities:
% DEFINE: ldent.
% USE: lden, lnbden.
% ....................................................................
%
ldent = lden + lnbden; % Net linear density including beam and non-beam.
%
% _____________________________________________________________________
% Section (4)
% Check & impose a valid number of FEM beam elements given the number
% of valid imposed nodes.  Also determine # of valid FEM nodes:
% DEFINE: nn.
% USE: nsel, nel.
% POSSIBLY ALTER: nel.
% ....................................................................
%
if nel < nsel;% If the # of desired elements is less than the # of super
  nel = nsel;%  elements imposed, match the # desired to the # imposed.
  warning = [int2str(nel) ...
          ' finite elements imposed between & around imposed nodes.']
end
nn = nel + 1; % number of total FEM nodes (after meshing).
%
warning = [];
% _____________________________________________________________________
% Section (5)
% First, I would like to create a vector of the cumulative norm of
% mass over stiffness.  The qualitative reasoning is that the portions of
% the beam with a large mass to stiffness ratio, should move about more
% and thus need more elements.  This metric can easily be changed.
% DEFINE: smperk.
% USE: ldent, lEI, ns.
% ....................................................................
%
smperk = cumsum(ldent./lEI); % Cumulative sum of mass per k.
% _____________________________________________________________________
% Section (6)
% Determine the number of free elements & normalize the metric.
% Since, I will have to insure that each major segment has at least
% one finite element, I will normalize the metric by the number
% of free nodes in Section 8. (Note: if no free nodes (nel==nbel),
% the metric is set to zeros in Section 8.)
% DEFINE: nelfr, nfm, nsmperk.
% USE: nsel, nel, smperk.
% ....................................................................
%
nelfr = nel - nsel; % This is the number of truly free elements.
%
nsmperk = smperk/max(smperk); % Normalized smperk from 0+ to nelfr
%
% _____________________________________________________________________
% Section (7)
% What I must now do is break-up the beam into the nsel super segments:
% For example, if nin = 2 (supports) one super segment before the first
% support, one between the supports, and one after the supports.  Then
% determine the number of free elements that would ideally exist in each
% super segment.  To achieve this I scale the metric to the number
% of free nodes and then ratchet it down, to spread out the free
% elements, and separately add in the imposed minimum of one finite
% element to each super segment.
% DEFINE: n.
% USE: nelfr, nsel, nsmperk, sv.
% ....................................................................
%
nf = zeros(nsel,1); % Initialize vector of integer approximation to the
%                   number of free elements per super segment.
%
for I = 1:nsel
  nf(I) = floor((nelfr+1)*nsmperk(sv(I+1))) - sum(nf); % Round scaled
  %                     cumulative metric to nearest integer, and
  %                     ratchet down by previous # of free elements
  %                     allocated to previous super segments.
end
nf(nsel) = nf(nsel) - 1;% The final super element picked-up an extra
% element since the normalization is based on nelfr+1 instead of nelfr.
% This normalization works because to place 'x' free elements evenly
% you cut the metric into 'x+1' pieces.  The last segments's last index
% just reaches 'x+1', so the final '1' must be removed.
%
n = ones(size(nf)) + nf;% This is a vector of the number of nodes per
%                   super segment. (Before Check.)
nelsv = diff(sv);% The number of elements per super segment.
if max(n./nelsv) > 1 % Check if more elements allotted than will fit.
  warning = ['While-loop entered to redistribute free elements from '...
          'supper elements that are too short to fit them all to '...
          'those that have room.']
  while max(n./nelsv) > 1
    [y, indthin] = max(n./nelsv);% Find index of s-element w/ too many.
    [y, indfat] = min(n./nelsv); % Find index of s-element w/ too few.
    n(indthin) = n(indthin) - 1; % Swap one element at a time.
    n(indfat) = n(indfat) + 1;
  end
end
nf = []; y = []; indthin = []; indfat = []; nelsv = []; warning = [];
% _____________________________________________________________________
% Section (8)
% What I must now do is partition the super segments, based on the
% total number of finite elements per segment.  To achieve this I
% will conduct an exercise similar to the one of section 7 for each
% super segment, excepting that I will identify the index directly.
% DEFINE: ncord.
% USE:  n, nsel, nn, ns, smperk, sv.
% ....................................................................
%
ncord = zeros(nn,1); % Initialize vector of all node points.
%
for I = 1:nsel
  smperki = smperk(sv(I):sv(I+1)) - smperk(sv(I));%sum M/K super segment I.
  nsmperki = smperki/max(smperki);% Normalized smperk over super segment I.
  nindi = zeros(n(I),1); % Initialize the node index over super segment I.
  %
  for j = 1:n(I)
    [y, nindi(j)] = min(abs(n(I)*nsmperki - (j-1)));% nindi is the relative
    %                                   index of the node in
    %                                   the super segment.
  end
  %
  if min(diff(nindi)) < 1 % Test for collocated nodes.
    warning = ['While-loop entered to redistribute collocated elements '...
            'from elements that are too short to fit them all to '...
            'those that have room.']
    nindi(find(diff(nindi) == 0)) = [];% Nullify collocated nodes.
    while (n(I) - length(nindi)) > 0
      [y, indfat] = max(diff(nindi));% Index of nindi w/ room for more.
      % Now, identify node between nindi(indfat) and indi(indfat+1).
      nindi = [nindi; round( mean(nindi([indfat (indfat+1)])) )];
      nindi = sort(nindi);
    end
  end
  %
  if I == 1
    np = 0;% Set-up previous index so that super segment elements
    %      begin at their first index, and end one before the following
    %      elements segment.  Setting np to 0 will facilitate
    %      the index starting at 1 for the first super segment.
  else
    np = sum(n(1:(I-1)));% Remaining previous indices sum up all
    %                   previous indices.
  end
  %
  ncord((np+1):(np+n(I))) = nindi + sv(I) - 1;% Recall, sv(I) is the absolute
  %                                   index of the start of the big
  %                                   element.
  %             (Since the lowest index nindi is 1, 1 must be subtracted.)
end
%
ncord(nn) = ns;% The final node point is imposed as the end of the beam.
if ncord(nn-1) == ns;% One last collocated node slipped by.
  ncord(nn-1) = [];% Nullify collocated node.
  [y, indfat] = max(diff(nindi));% Index of nindi w/ room for more.
  % Now, identify node between nindi(indfat) and indi(indfat+1).
  % Remember that sv(I) - 1 must be added.
  ncord = [ncord; round( mean(nindi([indfat (indfat+1)])) ) + sv(I) - 1];
  ncord = sort(ncord);
end

  ncord(nn) = ns;% The final node point is imposed as the end of the beam.
else

%
I = []; j = []; smperki = []; nsmperki = []; nindi = []; y = []; np = [];
indfat = [];
```

```
286 % _____
287 % Section (9) DISABLED (Addressed in elebeamint.m)
288 % Warn if finite element length is becoming short relative to geometric
289 % resolution.
290 % USE: ncord.
291 % .....
292 %
293 % if min(diff(ncord)) < 50
294 %  warning = ['Finite element length of ' int2str( min(diff(ncord))) ...
295 %  ' indices is small). Consider using fewer finite elements.'...
296 %  ' Also be sure that non-beam masses are distributed over '...
297 %  'neighboring indices in the lnbden vector.' ]
298 % end
299 % nn = nel + 1; % number of total FEM nodes (after meshing).
300 %
301 % warning = [];
302 % _____
303 % Section (10)
304 % If tlabel is set, plot the metric and node locations versus position in
305 % the current figure window.
306 % USE: nargin, ncord, nel, nsmperk, spatial, sv, tlabel.
307 % .....
308 %
309 if nargin == 7
310  clg
311  plot(spatial,nel*nsmperk,'k:')
312  hold on
313  stem(spatial(ncord),nel*nsmperk(ncord))
314  plot(spatial(sv),nel*nsmperk(sv),'k*')
315  grid
316  if isstr(tlabel); % Checks to see if tlabel is a string.
317   if strcmp(tlabel,'flag'); % Checks to see if tlabel is an old
318    tlabel = ['Beam']; %     convention.  If so, a default is applied.
319   elseif strcmp(tlabel,'tlabel'); % Checks to see if tlabel is used.
320    tlabel = ['Beam']; %            If so, a default is applied.
321   else
322    tlabel(1) = upper(tlabel(1)); % Imposes upper case on first letter.
323   end
324  else
325   tlabel = ['Beam']; % Default is applied in no valid tlabel given.
326  end
327  title([tlabel ' Meshing Metric & Node Locations Vs Position'])
328  xlabel('Position Along Beam')
329  ylabel('Normalized Metric {o"s->nodes, *"s->imposed nodes, :->metric}')
330  % Note: " provides for using ' within a 'string'.
331  hold off
332 end
333 % _____
334 % Completed:
335 % OUTPUT: ncord.
336 % .....
337 %
```

<fem_node_check.m>

```
1  function [ncord,spatial] = fem_node_check(ncord,spatial)
2  % fem_node_check.m
3  % [ncord] = fem_node_check(ncord);
4  % o  This m-file checks the consistency of the vector of node
5  %    finite element node coordinates, and spatial vector.
6  % o  This file uses:
7  %     geom_check.m in section 1
8  % o  Specific input/output variable definitions are:
9  %  ncord -> The vector of indices such that spatial(ncord) is the position
10 %     of each 2*(nel+1) generalized coordinates. (Also thought of as nodes.)
11 %  spatial -> Axial position vector, currently limited to equally spaced
12 %     position vectors with position data of every dx*n point such that
13 %     n = 1:length(spatial) and length(spatial)*dx equals max(spatial).
14 % o  Created 03 October 1995 by Eric Kathe.
15 %  Benet Labs, Watervliet Arsenal, NY 12189-4050 <ekathe@pica.army.mil>
16 % ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
17 % _____
18 % Section (1)
19 % A few checks to be sure input variables are the right size, et cetera.
20 % This section uses geom_check.m to validate the input geometry vectors.
21 % Also define the number of elements of spatial, ns.
22 % DEFINE: ns.
23 % USE: spatial.
24 % POSSIBLY ALTER: spatial.
25 % .....
26 %
27 ns = length(spatial); % number of elements of spatial.
28 %
29 [spatial] = geom_check(spatial);
30 % _____
31 % Section (2)
32 % Validate & sort node vector, ncord.
33 % USE: ncord, ns.
34 % POSSIBLY ALTER: ncord.
35 % .....
36 %
37 ncord = ncord(:);% Impose column structure.
38 ncord = sort(ncord);
39 nn = length(ncord);% # of nodes. (May be modified later.)
40 %
41 if sum(ceil(ncord)-ncord) > 0
42  ncord = ceil(ncord - ½);
43  warning = ['Non integer values of ncord encountered. ' ...
44        'values of ncord have been rounded to the nearest integer.']
45 end
46 if ncord(1) ~= 1
47  if ncord(1) < 1
48   ncord(1) = 1;
49   warning = ['Negative or zero node of ncord ignored. ' ...
50         'First node placed at index 1.  (The beginning of the beam)']
51  else
52   ncord = [1; ncord];
53   warning = ['First node of ncord must be at beginning of beam. ' ...
54         'First node at index 1 added to ncord.']
55   nn = nn + 1;% # of nodes increased to compensate for new node.
56   %        May still be modified later.
57  end
58 end
59 for I = 2:nn
60  if ncord(I) < 2
61   ncord(I) = 0;
62   warning = ['Negative, zero, or additional beginning of beam node ' ...
63         'of ncord ignored.']
64  elseif ncord(I) > ns
65   ncord(I) = 0;
66   warning = 'Node beyond beam in ncord ignored.'
67  elseif ncord(I) == ncord((I-1))
68   ncord(I) = 0;
69   warning = 'Repeated node of ncord ignored.'
70  end
71 end
72 %
73 ncord = sort(ncord(find(ncord)));% Exclude zero elements of ncord & resort.
74 %
75 nn = length(ncord);% # of nodes. (May be modified one more time.)
76 %
77 if ncord(nn) < ns
78  ncord = [ncord; ns];
79  warning = ['Last node of ncord must be at end of beam. ' ...
80        'End of beam index node added to ncord.']
81  nn = nn + 1;% Last possible modification to # of nodes.
82 end
83 %
84 I = []; nn = []; warning = [];
85 % _____
86 % Completed:
87 % OUTPUT: ncord, spatial.
88 % .....
89 %
```

<freq2str.m>

```
1  function t = freq2str(x, prec)
2  %NUM{FREQ}2STR Number to string conversion {For linear frequencies}.
3  % T = NUM{FREQ}2STR(X) converts the scalar number X into a string
4  % representation T with about 4 digits and an exponent if
5  % required.  This is useful for labeling plots with the
6  % TITLE, XLABEL, YLABEL, and TEXT commands.  An optional
7  % argument can be supplied for indicating an alternate precision.
8  % T = NUM{FREQ}2STR(X,PREC) converts the scalar {linear frequency}
9  %     number X into a string
10 % representation {using fixed representation} with a maximum
11 %     precision {number of digits to the right of the decimal point}
12 %     specified by PREC.
13 %
14 %     {Modified from MATLAB's num2str by Eric Kathe, 31 Oct 95.}
15 % See also INT2STR, SPRINTF, FPRINTF.
16
17 if isstr(x)
18   t = x;
19 else
20   if (nargin == 1)
21   num_format = '%.4g';
22   else
23   num_format = ['%.' num2str(prec) 'f'];% (Mod 1 of 2: 'g' to 'f'.)
24   end
25
26   t = sprintf(num_format, real(x));
27   if imag(x) > 0
```

```
      t = [t '+' sprintf(num_format, imag(x)) 'I'];
    elseif imag(x) < 0
      t = [t '-' sprintf(num_format, -imag(x)) 'I'];
    end
    t = [t ' Hz'];% (Mod 2 of 2: append ' Hz'.)
end
```

<geom_check.m>

```
function [spatial,pv1,pv2,pv3] = geom_check(spatial,pv1,pv2,pv3)
% geom_check.m
% [spatial,pv1,pv2,pv3] = geom_check(spatial,pv1,pv2,pv3)
% o  This m-file checks the geometric vector inputs for
%    later finite element formulation.
% o  The spatial vector must always be included. Up to three
%    additional vectors that are supposed to correspond one-to
%    one with spatial and have no negative elements can also be
%    validated.
% o  The input geometry vectors are generally entered and
%    computed using *_geomf.m.
% o  Specific input/output variable definitions are:
%  spatial -> Axial position vector, currently limited to equally spaced
%    position vectors with position data of every dx*n point such that
%    n = 1:length(spatial) and length(spatial)*dx equals max(spatial).
%  pv1,pv2,pv3 -> Vectors with no negative elements that correspond one
%    to one with the spatial vector. Common example for finite
%    element formulation include linear density and EI cross-sectional
%    property vectors versus spatial.
% o  Created 20 September - 05 October 1995 by Eric Kathe.
%  Benet Labs, Watervliet Arsenal, NY 12189-4050 <ekathe@pica.army.mil>
% ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
% _____________________________________________________________________________
% Section (1)
% Check spatial. Warning labels provide the reason for each if statement.
% DEFINE: eflag, ns.
% USE: spatial.
% POSSIBLY ALTER: spatial.
% .............................................................................
%
spatial = spatial(:); % Impose column structure.
ns = length(spatial); % Number of elements of spatial.
eflag = 0; % This will form a cumulative error count.
%
dspat = diff(spatial);
deltax = mean(diff(spatial));
if min(spatial) < deltax*(1 - 10^(-2))
  % Note: must allow for machine round off. Thus the 10^(-2) factor.
  spatial = spatial - min(spatial) + deltax;
  warning = ['Spatial must not be negative nor start at zero.'...
      ' It has been offset by the negative minimum '...
      'and it"s increment to make it"s first value start at '...
      'it"s incremental value. (Commonly 1mm.)']
  eflag = eflag + 1;
end
if abs(spatial(1) - deltax) > deltax*10^(-2)
  spatial = spatial - spatial(1) + deltax;
  warning = ['Spatial must start at it"s first incremental value.'...
        'It must not start at zero or some large positive value.'
        ' A DC offset has been applied to try to correct it.']
  eflag = eflag + 1;
end
%
if abs(max(dspat) - min(dspat)) > deltax*10^(-2)
  % Note: must allow for machine round off. Thus the 10^(-2) factor in
  % lieu of zero. (1% is also a reasonable variation.)
  spatial = max(spatial)*(1:ns)/ns;
  warning = 'Spatial must be evenly spaced. Even spacing has been imposed.'
  eflag = eflag + 1;
end
%
dspat = []; deltax = []; warning = [];

% _____________________________________________________________________________
% Section (2)
% Check pv1, pv2, pv3. Warning labels provide the reason for each if
% statement.
% USE: eflag, nargin, pv1, pv2, pv3.
% POSSIBLY ALTER: eflag, pv1, pv2, pv3,
% .............................................................................
%
for I = 1:(nargin - 1)
  eval(['pv = pv' int2str(I) ';']) % This assigns pvi to pv.
  %
  pv = pv(:); % Impose column structure.
  if length(pv) ~= ns
    dumv = zeros(ns,1);
    dumv(1:length(pv)) = pv;
    pv = dumv(1:ns);
    warning = ['Mismatched pv' int2str(I) ' size truncated '...
          'or padded to spatial size.']
    eflag = eflag + 1;
  end
  if min(pv) < 0; pv = abs(pv);
    warning = ['pv' int2str(I) ' must not be negative. It has '...
          'been reset to it"s absolute values.']
    eflag = eflag + 1;
  end
  eval(['pv' int2str(I) ' = pv;']) % This assigns pv to pvi.
end
%
dumv = []; I = []; pv = []; warning = [];
% _____________________________________________________________________________
% Section (3)
% Warn of Problems encountered:
% USE: eflag.
% .............................................................................
%
if eflag > 0
  warning = ['WARNING: ' int2str(eflag) ' serious error(s) in ' ...
  'geometry vectors encountered. Further results likely to be in error.']
end
%
warning = [];
% _____________________________________________________________________________
% Completed:
% OUTPUT: spatial, pv1, pv2, pv3. (If nargin >= 1,2,3,&4 respectively.)
% .............................................................................
%
```

<geom_nbseg.m>

```
function [indl,indr,lnbdenseg] = geom_nbseg(posl,posr,mass,spatial)
% geom_nbseg
% [lnbdenseg] = geom_nbseg(posl,posr,mass,spatial)
% o  This m-file converts information of a non beam mass, and converts
%    it to the lnbdenseg vector segment.
% o  This file uses:
%     geom_check.m
% o  Specific variable definitions are:
%  posl, posr -> The left and right indices of the desired segment in
%          the units of spatial. (Example: meters.)
%  mass -> The total non beam mass to be evenly distributed between
%       posl & posr in desired units. (Example: Kg.)
%  spatial -> Axial position vector, currently limited to equally spaced
%    position vectors with position data of every dx*n point such that
%    n = 1:length(spatial) and length(spatial)*dx equals max(spatial).
%  lnbdenseg -> A scaler of uniform linear density of the mass spread
%          out between the indices of posl & posr in spatial.
% o  Created 22 - 27 September 1995 by Eric Kathe.
%  Benet Labs, Watervliet Arsenal, NY 12189-4050 <ekathe@pica.army.mil>
% ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
% _____________________________________________________________________________
% Section (1)
% Check and validate input parameters. Warnings indicate the reason for
% the checks.
% This section uses geom_check.m
% USE: posl,posr,mass,spatial
% POSSIBLY ALTER: posl,posr,mass,spatial
% .............................................................................
%
%
[spatial] = geom_check(spatial); % Performs a series of checks to
%                   insure a valid spatial vector.
if posl > posr
  poss = posl;% Temporary swap value.
  posl = posr;
  posr = poss;
  warning = ['Right position should be larger than left. ' ...
        'Reset to (' num2str(posl) ',' num2str(posr) ').']
end
%
if posl < 0
  posl = 0;
  warning = ['Negative positions not valid. '...
        'Reset posl to (' num2str(posl) ').']
end
if posr < 0
  posr = 0;
  warning = ['Negative positions not valid. '...
        'Reset posr to (' num2str(posr) ').']
end
if posl > max(spatial)
  posl = max(spatial);
  warning = ['Positions greater than beam length are not valid. '...
```

```
        'Reset posl to (' num2str(posl) ').']
end
if posr > max(spatial)
  posr = max(spatial);
  warning = ['Positions greater than beam length are not valid. '...
        'Reset posr to (' num2str(posr) ').']
end
%
if mass < 0
  mass = 0;
  warning = ['Negative mass not valid. '...
        'Reset to ' num2str(mass) '.']
end
%
poss = []; warning = [];
% ____________________________________________________________________
% Section (2)
% Find the indices of desired locations in spatial, and evaluate the
% length of the segment over which the mass is to be applied.
% DEFINE: indl, indr, seglen.
% USE: posl, posr, spatial.
% ..................................................................
%
[y, indl] = min(abs(spatial - posl)); % Find the left index.
[y, indr] = min(abs(spatial - posr)); % Find the right index.
%
deltax = mean(diff(spatial));% m (The delta x of spatial.)
%
% Determine total segment length in spatial. This length may
% vary slightly due to finding the closest index matches (indl & indr)
% above. (Note that the deltax is required to add in the length of
% spatial(indl) as in one foot equals 12" - 1" + 1".)
%
seglen = spatial(indr) - spatial(indl) + deltax;
%
deltax = []; y = [];
% ____________________________________________________________________
% Section (3)
% Evaluate the linear density.
% DEFINE: lnbdenseg.
% USE: mass, seglen.
% ..................................................................
%
lnbdenseg = mass/seglen; % Determine linear density = m/L.
%
% ____________________________________________________________________
% Completed:
% OUTPUT: indl, indr, lnbdenseg.
% ..................................................................
%
```

<geom_seg.m>

```
function [tseg] = geom_seg(indl,indr,tl,tr)
% geom_seg
% [tseg] = geom_seg(indl,indr,tl,tr)
% o  This m-file converts raw information of a beam segment, read off
%    of a drawing, to the geometry vector segment. It determines
%    from the number of input arguments if the segment is
%    uniform or consists of a linear taper.
% o  Specific variable definitions are:
%    indl, indr -> The left and right indices of the desired segment.
%    tl,tr -> The width or thickness of the segment at the left and right
%          indices respectively. If no tr given, a uniform segment
%          is assumed.
%    tseg -> A vector of length 1:(indr-indl+1) that represents the tapered
%          or uniform thickness of the segment.
% o  Created 22 September 1995 by Eric Kathe.
%    Benet Labs, Watervliet Arsenal, NY 12189-4050 <ekathe@pica.army.mil>
% ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
% ____________________________________________________________________
% Section (1)
% Check and validate input parameters. Warnings indicate the reason for
% the checks.
% DEFINE: nargflag.
% USE: indl, indr, nargin, tl, tr.
% POSSIBLY ALTER: indl, indr, tl, tr.
% ..................................................................
%
nargflag = nargin;% This is the number of input arguments.
%
indv = [indl indr]';
if sum(ceil(indv) - indv) ~= 0
  indv = ceil(indv - ½);
  warning = ['Indices must be integer. '...
        'Reset to (' int2str(indv(1)) ',' int2str(indv(2)) ').']
end
if indv(1) > indv(2)
  indv = sort(indv);
  warning = ['right index should be larger than left. '...
        'Reset to (' int2str(indv(1)) ',' int2str(indv(2)) ').']
end
if min(indv) < 1
  indv = indv - min(indv) + 1;
  warning = ['Negative and zero indices not valid. '...
        'Reset to (' int2str(indv(1)) ',' int2str(indv(2)) ').']
end
indl = indv(1);
indr = indv(2);
%
if tl < 0
  tl = 0;
  warning = ['Negative tl not valid. '...
        'Reset to ' int2str(tl) '.']
end
if nargflag == 4;% Indicates that tr defined.
  if tr < 0
    tr = 0;
    warning = ['Negative tr not valid. '...
        'Reset to ' int2str(tr) '.']
  end
  if tr == tl;% If both thicknesses are equal, compute as non-tapered.
    nargflag = 3;
  end
end
%
indv = []; warning = [];
% ____________________________________________________________________
% Section (2)
% If the section is not tapered, evaluate tseg directly.
% Else, compute as tapered.
% DEFINE: tseg.
% USE: indl, indr, nargflag, tl, tr.
% ..................................................................
%
if nargflag == 3
  tseg = tl*ones(((indr - indl)+1),1);
else
  tap = (0:(indr - indl))';
  ntap = tap/max(tap); % Linear vector from zero to one.
  tseg = tl*ones(size(ntap)) - (tl-tr)*ntap;% Tl minus the linear
  %                    taper difference (tl-tr)*tap to tr.
end
%
ntap = []; tap = [];
% ____________________________________________________________________
% Completed:
% OUTPUT: tseg.
% ..................................................................
%
```

<geomf_XM291.m>

```
function [spatial, lden, lEI, lnbden, Mextl, Mextr, gm] = ...
                    geomf_XM291(tlabel)
% geomf_XM291.m
% [spatial, lden, lEI, lnbden, Mextl, Mextr, gm] = geomf_XM291(tlabel);
% o  This m-file generates the linear density and EI for the XM291
%    gun system.
% o  The geometric information contained was derived from XM291 Drawings
%    # WTV-F37060 sheets 1-4 of 4.
% o  This file uses:
%       geom_seg.m
%       geom_nbseg.m
%       geom_check.m
% o  Specific variable definitions are:
%    tlabel -> Enables plotting of the beam geometry.
%      If valid, the string of tlabel is incorporated into the plot to
%      describe the beam.
%    spatial -> Axial position vector, currently limited to equally spaced
%      position vectors with position data of every dx*n point such that
%      n = 1:length(spatial) and length(spatial)*dx equals max(spatial).
%    lden -> Beam linear density vector such that each index value
%          corresponds to the beam position of the respective spatial
%          value at the same index.
%    lEI -> Similar to lden except for linear EI cross-section properties.
%    lnbden -> Similar to lden, except this records the inertia of non-beam
%          masses that are attached to the beam.
%    Mextl, Mextr -> 2x2 sub matrices of left and right extreme rigid body
%          inertia. This special treatment of a non-beam mass is
%          required when the location is beyond the barrel. (It
%          will later be added to the M & K matrices along with
%          the constraint forces.)
%    gm -> Similar to lden except the columns of this matrix record the
```

```
%       inner and outer radii respectively.
% o  Created 30 August - 28 November 1995 by Eric Kathe.
%    Benet Labs, Watervliet Arsenal, NY  12189-4050 <ekathe@pica.army.mil>
% ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
% ________________________________________________________________________
% Section (1)
% The approach here will be to record the outer radii of the barrel in
% millimeters per each millimeter.  When a taper is encountered,
% a vector decrement method will be used.  Indl & indr define the extreme
% indices of the segment.  For example, 19 to 75 indicates radii beginning
% at axial position 0.019m and ending at 0.075m.  Diameters are divided by
% two.
% (Note: The semantics used in the descriptions are not necessarily those
%       used by the gun designers.  Refer to the drawings based on index
%       location to avoid confusion.)
% This section uses geom_seg.m.
% DEFINE: rout.
% USE: author provided data as listed.
% ........................................................................
%
rout = zeros(6750,1); % Initialize rout vector.
%
% A) Flat prior to rear threads. {sheet 2, section N-N & e4}:
indl = 1;
indr = 37;
rL = 291.84/2;
rout(indl:indr) = geom_seg(indl,indr,rL);
%
% B) Rear threads. {sheet 2, section N-N, D between Major & Minor, & b2}:
indl = 38;
indr = 211;
rL = 298.24/2;
rout(indl:indr) = geom_seg(indl,indr,rL);
%
% C) Flat following treads. {sheet 2, b2}:
indl = 212;
indr = 224;
rL = 291.84/2;
rout(indl:indr) = geom_seg(indl,indr,rL);
%
% D) Taper prior to trunnion. {sheet 1, e8-7}:
indl = 225;
indr = 1100;
rL = 305/2;
rR = 304.8/2;
rout(indl:indr) = geom_seg(indl,indr,rL,rR);
%
% E) Flat at trunnion. {sheet 1, section EE & e6}:
indl = 1101;
indr = 1323;
rL = 304.8/2;
rout(indl:indr) = geom_seg(indl,indr,rL);
%
% F) Lip after trunnion flat. {sheet 1, e7}:
indl = 1324;
indr = 1332;
rL = 304.8/2;
rR = 287/2;
rout(indl:indr) = geom_seg(indl,indr,rL,rR);
%
% G) Flat forward of trunnion. {sheet 1, e6 d6}:
indl = 1333;
indr = 1428;
rL = 287/2;
rout(indl:indr) = geom_seg(indl,indr,rL);
%
% H) Taper forward of trunnion. {sheet 1, e6}:
indl = 1429;
indr = 1603;
rL = 287/2;
rR = 260/2;
rout(indl:indr) = geom_seg(indl,indr,rL,rR);
%
% I) The next Flat. {sheet 1, e6 d6}:
indl = 1604;
indr = 1661;
rL = 260/2;
rout(indl:indr) = geom_seg(indl,indr,rL);
%
% J) Second lip after trunnion flat. {sheet 1, e7 detail A-A}:
indl = 1662;
indr = 1663;
rL = 260/2;
rR = 258/2;
rout(indl:indr) = geom_seg(indl,indr,rL,rR);
%
% K) First long taper. {sheet 1, e6-5}:
indl = 1664;
indr = 3041;
rL = 258/2;
rR = 212.60/2;
rout(indl:indr) = geom_seg(indl,indr,rL,rR);
%
% L) Flat prior to bore evacuator. {sheet 1, e5}:
indl = 3042;
indr = 3102;
rL = 220/2;
rout(indl:indr) = geom_seg(indl,indr,rL);
%
% M) First taper of bore evacuator. {sheet 1, e5}:
indl = 3103;
indr = 3112;
rL = 220/2;
rR = 210/2;
rout(indl:indr) = geom_seg(indl,indr,rL,rR);
%
% N) Second flat prior to bore evacuator. {sheet 1, e5}:
indl = 3113;
indr = 3165;
rL = 215/2;
rout(indl:indr) = geom_seg(indl,indr,rL);
%
% O) Thread flat prior to bore evacuator. {sheet 1, e5}:
indl = 3166;
indr = 3200;
rL = 207.80/2;
rout(indl:indr) = geom_seg(indl,indr,rL);
%
% P) Main taper of bore evacuator. {sheet 1, e5-4}:
indl = 3201;
indr = 3902;
rL = 207.80/2;
rR = 184/2;
rout(indl:indr) = geom_seg(indl,indr,rL,rR);
%
% Q) First flat after bore evacuator. {sheet 1, e4}:
indl = 3903;
indr = 3954;
rL = 190/2;
rout(indl:indr) = geom_seg(indl,indr,rL);
%
% R) Second flat after bore evacuator. {sheet 1, e4}:
indl = 3955;
indr = 4183;
rL = 183/2;
rout(indl:indr) = geom_seg(indl,indr,rL);
%
% S) Third flat after bore evacuator. {sheet 1, e4}:
indl = 4184;
indr = 4234;
rL = 177.40/2;
rout(indl:indr) = geom_seg(indl,indr,rL);
%
% T) Second long taper. {sheet 1, e4-2}:
indl = 4235;
indr = 6403;
rL = 177.40/2;
rR = 156.60/2;
rout(indl:indr) = geom_seg(indl,indr,rL,rR);
%
% U) Taper prior to muzzle. {sheet 1, e1}:
indl = 6404;
indr = 6428;
rL = 156.60/2;
rR = 169.8/2;
rout(indl:indr) = geom_seg(indl,indr,rL,rR);
%
% V) First flat of muzzle. {sheet 4, f7-6}:
indl = 6429;
indr = 6527;
rL = 169.8/2;
rout(indl:indr) = geom_seg(indl,indr,rL);
%
% W) First groove of muzzle. {sheet 4, f7-6}:
indl = 6528;
indr = 6533;
rL = 159/2;
rout(indl:indr) = geom_seg(indl,indr,rL);
%
% X) Taper between grooves of muzzle. {sheet 1, e1}:
indl = 6534;
indr = 6573;
```

```
rL = 165.313/2;
rR = 164/2;
rout(indl:indr) = geom_seg(indl,indr,rL,rR);
%
% Y) Second flat of muzzle. {sheet 4, f6-5}:
indl = 6574;
indr = 6644;
rL = 164/2;
rout(indl:indr) = geom_seg(indl,indr,rL);
%
% Z) Second groove of muzzle. {sheet 4, f5}:
indl = 6645;
indr = 6650;
rL = 157.5/2;
rout(indl:indr) = geom_seg(indl,indr,rL);
%
% AA) Threads of muzzle. {sheet 4, f5}:
indl = 6651;
indr = 6673;
rL = 6.3037*(25.4)/2;% English thread: (in)(mm/in)
rout(indl:indr) = geom_seg(indl,indr,rL);
%
% AB) The final flat. {sheet 4, e5-4}:
indl = 6674;
indr = 6750;
rL = 157.5/2;
rout(indl:indr) = geom_seg(indl,indr,rL);
%
indl = []; indr = []; rL = []; rR = [];
% _____
% Section (2)
% The approach here will be to record the inner diameters of the barrel in
% analogy with section 1.
% (Note: The semantics used in the descriptions are not necessarily those
%       used by the gun designers. Refer to the drawings based on index
%       location to avoid confusion.)
% DEFINE: rin.
% USE: rout, author provided data as listed.
% .....
%
rin = zeros(size(rout)); % Initialize rin vector.
%
% A) Chamfer taper. {sheet 3, f6}:
indl = 1;
indr = 30;
rL = 159.0951/2;
rR = 158.36/2;
rin(indl:indr) = geom_seg(indl,indr,rL,rR);
%
% B) Gas check seal. {sheet 3, f6}:
indl = 31;
indr = 61;
rL = 158.36/2;
rR = 157.6/2;
rin(indl:indr) = geom_seg(indl,indr,rL,rR);
%
% C) Flat of chamber. {sheet 3, f5-6}:
indl = 62;
indr = 487;
rL = 157.6/2;
rin(indl:indr) = geom_seg(indl,indr,rL);
%
% D) First taper of chambrage cone. {sheet 3, f4}:
indl = 488;
indr = 515;
rL = 157.6/2;
rR = 140.14/2;
rin(indl:indr) = geom_seg(indl,indr,rL,rR);
%
% E) Second taper of chambrage cone. {sheet 3, f4}:
indl = 516;
indr = 555;
rL = 140.14/2;
rR = 121.84/2;
rin(indl:indr) = geom_seg(indl,indr,rL,rR);
%
% F) Taper of forcing cone. {sheet 3, f4}
indl = 556;
indr = 602;
rL = 121.84/2;
rR = 120.85/2;
rin(indl:indr) = geom_seg(indl,indr,rL,rR);
%
% G) The barrel. {sheet 3, f4-3}:
indl = 603;
indr = 6750;
rL = 120.85/2;
rin(indl:indr) = geom_seg(indl,indr,rL);
%
indl = []; indr = []; rL = []; rR = [];
% _____
% Section (3)
% Now to convert units to meters, and create a corresponding position
% vector in meters with indices corresponding to the rin & rout vectors.
% DEFINE: ns, spatial.
% USE: rin, rout.
% Alter: rin, rout.
% .....
%
rout = rout/1000;% mm/(mm/m)
rin = rin/1000;% mm/(mm/m)
ns = length(rout);% Number of geometric vector indices.
spatial = (1:ns)'/(1000);% (mm)/(mm/m)
% _____
% Section (4)
% Now compute the cross-sectional properties of the beam.
% Also compute the total mass to validate the geometry.
% DEFINE: deltax, lden, lEI, mass, masseng.
% USE: rin, rout, spatial, author provided data as listed.
% .....
%
deltax = mean(diff(spatial));% m (The disk thickness.)
density = 489*(0.4536)/(0.028317);% lbm/ft3(Kg/lbm)/(m3/ft3)
%       MARK's 9th, table page 6.44, Carbon steel (0.40% C).
%       Gun steel 4337m close to 4340 w/ some extra Vanadium
%       added for machinability. (4340 -> .38 - .43 % C)
lden = density*pi*(rout.^2 - rin.^2);% (Kg/m^3)(m^2)
%
mass = sum(lden)*deltax; % (Kg/m)(m)
masseng = mass*(1/0.4536);% Kg(1/(Kg/lbm))
%
E = (29.5*10^6)*6894.8; % lbf/in2*((N/m2)/(lbf/in2))
%  Private communication with Dr. Ron Gast, AMATA-AR-CCB-DE.
%  (Shingley & Mitchell, 4th, table A-18 E's for steel from
%   29-30 Mpsi.) (Very heat treat dependent.)
%
lEI = E*(rout.^4 - rin.^4)*pi/4;
%
density = []; E = [];
% _____
% Section (5)
% Add in the external rigid mass. This mass is treated differently than
% simple non-beam mass (to be computed in section 6) due to the
% location of its center of gravity beyond the end of the beam. Therefore,
% it will be treated as a lumped inertia and directly coupled to the
% generalized coordinates of the first & or last node at a later time. It
% is important to note that this formulation provides for the inclusion of
% rigid body rotational inertia and could be used within the beam for non-
% beam masses where the linear density approximation is poor.
% For the case of a left external mass (Note: r would be negative):
%  x(ext) = x(1) + r*theta(1) -> dx(ext) = dx(1) + r*dtheta(1)
%  ddx(ext) = ddx(1) + r*ddtheta(1), ddtheta(ext) = ddtheta(1)
%  F(1) = mbr*ddx(ext), M(1) = (mext*r^2 + Jbr)*ddtheta(ext)
%  [m11 m12; m21 m22]*[ddx(1) ddtheta(1)]' = [F(1) M(1)]'
%
% DEFINE: ldenextl, ldenextr, Mextl, Mextr, spextl, spextr, massext.
% USE: deltax, author provided data as listed.
% .....
%
massext = 0;
%
% A) Left external mass: The Breech, mb = 1300#, located 10.9" RFB ->
% -0.6 RFT. (RFB & RFT are the rear face of breech and tube respectively.)
% For the breech, quick measurements indicate the breach is about 21"
% axial, and 15" high, for a J of m(1/12)bh(b^2+h^2)/(bh).
%  x(ext) = x(1) + r*theta(1) -> dx(ext) = dx(1) + r*dtheta(1)
%  ddx(ext) = ddx(1) + r*ddtheta(1), ddtheta(ext) = ddtheta(1)
%  F(1) = mbr*ddx(ext), M(1) = (mext*r^2 + Jbr)*ddtheta(ext)
%  [m11 m12; m21 m22]*[ddx(1) ddtheta(1)]' = [F(1) M(1)]'
mext = 1300*(0.4536);% lbm*(Kg/lbm)
rext = -0.6*(0.0254);% in*(m/in)
bext = 21*(0.0254);% in*(m/in)
hext = 15*(0.0254);% in*(m/in)
Jext = mext*(1/12)*(bext^2+hext^2);
%
Mextl = [mext mext*rext; mext*rext (Jext + mext*rext^2)];
%
rextl = (rext - bext/2);% in*(cm/in)*(m/cm)
iextl = floor(rextl/deltax + ½);% Round to nearest increment of deltax.
rextr = (rext + bext/2);% in*(cm/in)*(m/cm)
iextr = floor(rextr/deltax + ½);
spextl = deltax*(iextl:iextr)';
```

```
ldenextl = mextl/(max(spextl) - min(spextl))*ones(size(spextl));
massext = massext + mext;
%
% B) Right external mass: None.
Mextr = zeros(2,2);
spextr = [];
ldenextr = [];
massext = massext + 0;
%
mext = []; rext = []; bext = []; hext = []; iextl = []; iextr = [];
Jext = []; rextl = []; rextr = [];
% ____________________________________________________________
% Section (6)
% Add in non-beam masses. (Since this data is not often directly on
% drawings, the nearest indices of spatial are found from positions
% expressed in general units of measure, rather than indices directly.)
% Also compute the total non-beam mass to validate the geometry.
% This section uses geom_nbseg.m.
% DEFINE: lnbden, massnb.
% USE: deltax, ns, spatial, author provided data as listed.
% ..........................................................
%
lnbden = zeros(ns,1); % Initialize lnbden vector.
%
%
% A) The cradle, mcr = 1000#, 46.26 RFB -> 34.76 RFT.
% (RFB & RFT are the rear face of breech and tube respectively.)
% Since this mass is clearly distributed, I'll place over a 40" span
% centered at the center of gravity. (Measured distribution ~ 40".)
%
massnb = 1000*(0.4536);% lbm*(Kg/lbm)
poscg = 34.76*(0.0254);% in*(m/in) Position of center of gravity.
span = 40*(0.0254);% in*(m/in) Span of distribution.
% Note: poscg +- span/2 must lay within the beam. If they don't,
% geom_nbseg will apply the closest valid value and warn of change.
%
posl = poscg - span/2;% Left most position of distribution.
posr = poscg + span/2;% Right most position of distribution.
%
% Identify indices of lennb and effective linear density of mnb.
[indl,indr,lnbdenseg] = geom_nbseg(posl,posr,massnb,spatial);
%
% The non-beam mass is now added into the lnbden vector:
lnbden(indl:indr) = lnbden(indl:indr) + lnbdenseg;
%
% B) The recoil brakes (all 4): mrb = 528# @ 37"RFB -> 25.5"RFT.
% Since this mass is clearly distributed, I'll place over a 30"
% span centered at the CG. (measured distribution ~ 30".)
%
massnb = 528*(0.4536);% lbm*(Kg/lbm)
poscg = 25.5*(0.0254);% in*(m/in)
span = 30*(0.0254);% in*(m/in)
posl = poscg - span/2;% Left most position of distribution.
posr = poscg + span/2;% Right most position of distribution.
[indl,indr,lnbdenseg] = geom_nbseg(posl,posr,massnb,spatial);
lnbden(indl:indr) = lnbden(indl:indr) + lnbdenseg;
%
% C) The recuperators: mrc = 80# @ 36.5"RFB -> 25.0"RFT.
% Since this mass is clearly distributed, I'll place over a 20"
% span centered at the CG.
%
massnb = 80*(0.4536);% lbm*(Kg/lbm)
poscg = 25.0*(0.0254);% in*(m/in)
span = 20*(0.0254);% in*(m/in)
posl = poscg - span/2;% Left most position of distribution.
posr = poscg + span/2;% Right most position of distribution.
[indl,indr,lnbdenseg] = geom_nbseg(posl,posr,massnb,spatial);
lnbden(indl:indr) = lnbden(indl:indr) + lnbdenseg;
%
% D) The hydraulic manifold: mhm = 70# @ 31"RFB -> 19.5"RFT.
% Since this mass is clearly distributed, I'll place over a 15"
% span centered at the CG.
%
massnb = 70*(0.4536);% lbm*(Kg/lbm)
poscg = 19.5*(0.0254);% in*(m/in)
span = 15*(0.0254);% in*(m/in)
posl = poscg - span/2;% Left most position of distribution.
posr = poscg + span/2;% Right most position of distribution.
[indl,indr,lnbdenseg] = geom_nbseg(posl,posr,massnb,spatial);
lnbden(indl:indr) = lnbden(indl:indr) + lnbdenseg;
%
% E) The rails & yoke: mry = 200# @ 50.4"RFB -> 38.9"RFT.
% Since this mass is clearly distributed, I'll place over a 10"
% span centered at the CG.
%
massnb = 200*(0.4536);% lbm*(Kg/lbm)
poscg = 38.9*(0.0254);% in*(m/in)
span = 10*(0.0254);% in*(m/in)
posl = poscg - span/2;% Left most position of distribution.
posr = poscg + span/2;% Right most position of distribution.
[indl,indr,lnbdenseg] = geom_nbseg(posl,posr,massnb,spatial);
lnbden(indl:indr) = lnbden(indl:indr) + lnbdenseg;
%
% F) DISABLED: (Not mounted for testing.)
% The bore evacuator, 67# @ 3.200m to 3.902m
% Since this mass is clearly distributed, I'll place over the span
% 3.200m to 3.902m as indicated on sheet 1 of 4 of the XM291 drawings.
%
% massnb = 67*(0.4536);% lbm*(Kg/lbm)
% posl = 3.200;% Left most position of distribution.
% posr = 3.902;% Right most position of distribution.
% [indl,indr,lnbdenseg] = geom_nbseg(posl,posr,massnb,spatial);
% lnbden(indl:indr) = lnbden(indl:indr) + lnbdenseg;
%
% G) DISABLED: (Not mounted for testing.)
% The muzzle reference mount, 15# @ 264.6" RFT. About 1.5" wide.
%
% massnb = 15*(0.4536);% lbm*(Kg/lbm)
% poscg = 264.6*(0.0254);% in*(m/in)
% span = 1.5*(0.0254);% in*(m/in)
% posl = poscg - span/2;% Left most position of distribution.
% posr = poscg + span/2;% Right most position of distribution.
% [indl,indr,lnbdenseg] = geom_nbseg(posl,posr,massnb,spatial);
% lnbden(indl:indr) = lnbden(indl:indr) + lnbdenseg;
%
% H) No additional non-beam masses. Sum up total mass.
%
massnb = sum(lnbden)*deltax;
%
%
indl = []; indr = []; lnbdenseg = []; poscg = []; posl = [];
posr = []; span = [];
% ____________________________________________________________
% Section (7)
% Define the geometry matrix. (Used by later files for plotting.)
% DEFINE: gm.
% USE: rin, rout.
% ..........................................................
%
gm = [rin rout];
% ____________________________________________________________
% Section (8)
% If flag is set, plot the property vectors and geometric matrix for
% visual inspection in the current figure window. (Note: in order to
% preserve the visual inspection, the non-beam masses are plotted
% within the confines of the geometry matrix. Otherwise the visual
% validation of linear beam density could be obscured.)
% USE: deltax, lEl, gm, lden, lnbden, ldenextl, ldenextr, ns, spatial,
%      spextl, spextr, tlabel.
% ..........................................................
%
if nargin == 1
  clg
  %
  % A) Determine if tlabel is a valid title string or apply default:
  %
  if isstr(tlabel); % Checks to see if tlabel is a string.
    tlabel(1) = upper(tlabel(1)); % Imposes uppercase on first letter.
  else
    tlabel = ['XM291 Radial'];% Default is used if no valid tlabel given.
  end
  %
  % B) Combine external and non beam masses into one vector:
  %
  nbxlden = lnbden;% Initialize the combined external and non beam
  %     mass vector that is within the confines of the beam geometry.
  %
  lobindx = find(spextl < deltax);% Left of beam index.
  lobspat = spextl(lobindx);% Left of beam spatial vector.
  loblden = ldenextl(lobindx);% Left of beam linear density.
  coindx = find(spextl >= deltax);% Index of ldenextl within beam.
  spatindx = (1:length(coindx))';% Indices of ldenextl in terms of
  %                              spatial, within beam.
  % Now combine the left external and non beam linear density that
  % coexists within the confines of the beam geometry.
  nbxlden(spatindx) = nbxlden(spatindx) + ldenextl(coindx);
  %
  robindx = find(spextr > max(spatial));% Right of beam index.
  robspat = spextr(robindx);% Right of beam spatial vector.
  roblden = ldenextr(robindx);% Right of beam linear density.
  coindx = find(spextr <= max(spatial));% Index of ldenextr within beam.
  spatindx = ((ns-length(coindx)):ns)';% Indices of ldenextr in terms of
```

```
548  %                    spatial, within beam.
549  % Now combine the right external and non beam linear density that
550  % coexists within the confines of the beam geometry.
551  nbxlden(spatindx) = nbxlden(spatindx) + ldenextr(coindx);
552  %
553  % C) Determine plotting scale to force external and non beam densities
554  %   to be plotted within the confines of the beam geometry.
555  %
556  nbi = find(nbxlden);% Find external & non beam density index locus.
557  psf = (0.50)*min( min(gm(nbi,:)')./nbxlden(nbi) );% Plot scale factor.
558  %
559  % D) Combine external and non beam mass totals:
560  %
561  nbmass = sum(lnbden)*dellax + massext; % (Kg/m)(m)+(Kg) Nbeam&ext mass.
562  nbmasseng = nbmass*(1/0.4536); % (Kg)(1/(Kg/lbm))
563  %
564  % E) Plot results:
565  %
566  subplot(311), plot(spatial,gm,'k'); hold on
567   plot(spatial,-gm,'k');
568   stem(spatial(nbi),psf*nbxlden(nbi));
569   stem(spatial(nbi),-psf*nbxlden(nbi));
570   stem(lobspat,psf*loblden); stem(lobspat,-psf*loblden);
571   stem(robspat,psf*roblden); stem(robspat,-psf*roblden);
572   hold off
573   tstring = [tlabel ' Profile & Non beam Masses Versus Length '...
574         'in Meters, (Total Non beam Mass of ' ...
575          num2str(nbmass) ' Kg, or ' num2str(nbmasseng) ' lbm)'];
576   title(tstring)
577   ylabel('m')
578   axis([max(spatial)*[-0.1 1.1], max(rout)*[-1.2 1.2]]); grid;
579   %
580   % The following axis manipulation scales the subplot for better printing.
581   pos = get(gca,'position'); % This is in normalized coordinates
582   pos(4)=pos(4)*.85;      % Shrink the height by a factor of .85
583   pos(2)=pos(2) + pos(4)*.15;% Raise the subplot by the saved height.
584   set(gca,'position',pos);
585   %
586  subplot(312), plot(spatial,lden,'k');
587   tstring = ['Linear Density from Profile, (Total Mass of Beam ' ...
588         num2str(mass) ' Kg, or ' num2str(masseng) ' lbm)'];
589   title(tstring)
590   ylabel('Kg/m')
591   axis([max(spatial)*[-0.1 1.1], max(lden)*[0 1.2]]); grid;
592   %
593   % The following axis manipulation scales the subplot for better printing.
594   pos = get(gca,'position'); % This is in normalized coordinates
595   pos(4)=pos(4)*.85;      % Shrink the height by a factor of .85
596   pos(2)=pos(2) + pos(4)*.15;% Raise the subplot by the saved height.
597   set(gca,'position',pos);
598   %
599  subplot(313), plot(spatial,lEI,'k');
600   ylabel('N-m^2')
601   xlabel('Axial Position (m)')
602   title('EI from Profile')
603   axis([max(spatial)*[-0.1 1.1], max(lEI)*[0 1.2]]); grid;
604   %
605   % The following axis manipulation scales the subplot for better printing.
606   pos = get(gca,'position'); % This is in normalized coordinates
607   pos(4)=pos(4)*.85;      % Shrink the height by a factor of .85
608   pos(2)=pos(2) + pos(4)*.15;% Raise the subplot by the saved height.
609   set(gca,'position',pos);
610   %
611 end
612 %
613 coindx = []; lobindx = []; loblden = []; lobspat = []; nbi = [];
614 nbmass = []; nbmasseng = [];  nbx = []; den = []; pos = []; psf = [];
615 robindx = []; roblden = []; robspat = []; spatindx = []; tstring = [];
616 % _____________________________________________________________________
617 % Completed:
618 % OUTPUT: gm, lden, lEI, lnbden, Mextl, Mextr, spatial.
619 % .....................................................................
620 %
```

<geomf_hybrid.m>

```
1   function [spatial, lden, lEI, lnbden, Mextl, Mextr, gm] = geomf_hybrid(tlabel)
2   % geomf_hybrid.m
3   % [spatial, lden, lEI, lnbden, Mextl, Mextr, gm] = geomf_hybrid(tlabel);
4   % o   This m-file generates the linear density and EI for Ron Gast's 60mm
5   %     gun system.
6   % o   The geometric information contained was derived from Drawings
7   %     # WTV-F34406 Sheets 1 & 2.
8   % o   This file uses:
9   %       geom_seg.m
10  %       geom_nbseg.m
11  %         geom_check.m
12  % o   Specific variable definitions are:
13  %   tlabel -> Enables plotting of the beam geometry.
14  %     If valid, the string of tlabel is incorporated into the plot to
15  %     describe the beam.
16  %   spatial -> Axial position vector, currently limited to equally spaced
17  %     position vectors with position data of every dx*n point such that
18  %     n = 1:length(spatial) and length(spatial)*dx equals max(spatial).
19  %   lden -> Beam linear density vector such that each index value
20  %     corresponds to the beam position of the respective spatial value
21  %     at the same index.
22  %   lEI -> Similar to lden except for linear EI cross-section properties.
23  %   lnbden -> Similar to lden, except this records the inertia of non-beam
24  %     masses that are attached to the beam.
25  %   gm -> Similar to lden except the columns of this matrix record the
26  %       inner and outer radii respectively.
27  % o   Created 12 September - 05 October 1995 by Eric Kathe.
28  %   Benet Labs, Watervliet Arsenal, NY  12189-4050 <ekathe@pica.army.mil>
29  % ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
30  % _____________________________________________________________________
31  % Section (1)
32  % The approach here will be to record the outer radii of the barrel in
33  % inches per each 10 mils (ie, 0.01 in).  When a taper is encountered,
34  % a vector decrement method will be used.  Indl & indr define the extreme
35  % indices of the segment.  For example, 19 to 75 indicates radii beginning
36  % at axial position 0.19" and ending at 0.75."  Diameters are divided by
37  % two.
38  % This section uses geom_seg.m.
39  % DEFINE: rout.
40  % USE: author provided data as listed.
41  % .....................................................................
42  %
43  rout = zeros(14500,1); % Initialize rout vector.
44  %
45  % A) Chamfer prior to rear threads. {sheet 2, b7}:
46  indl = 1;
47  indr = 18;
48  rL = 4.430/2;
49  rR = 4.466/2;
50  rout(indl:indr) = geom_seg(indl,indr,rL,rR);
51  %
52  % B) Flat prior to rear threads. {sheet 1, g8}:
53  indl = 19;
54  indr = 75;
55  rL = 4.466/2;
56  rout(indl:indr) = geom_seg(indl,indr,rL);
57  %
58  % C) Threads. {sheet 1, g7-8}:
59  indl = 76;
60  indr = 325;
61  rL = 4.670/2;
62  rout(indl:indr) = geom_seg(indl,indr,rL);
63  %
64  % D) Groove after threads. {sheet 1, g7}:
65  indl = 326;
66  indr = 356;
67  rL = 4.47/2;
68  rout(indl:indr) = geom_seg(indl,indr,rL);
69  %
70  % E) Second chamfer.  {sheet 1, g7}:
71  indl = 357;
72  indr = 368;
73  rL = 4.756/2;
74  rR = 4.996/2;
75  rout(indl:indr) = geom_seg(indl,indr,rL,rR);
76  %
77  % F) Forward flat of breech area. {sheet 1, g7}:
78  indl = 369;
79  indr = 500;
80  rL = 4.996/2;
81  rout(indl:indr) = geom_seg(indl,indr,rL);
82  %
83  % G) First major segment. {sheet 1, c-d,7-5}:
84  indl = 501;
85  indr = 3000;
86  rL = 5.40/2;
87  rout(indl:indr) = geom_seg(indl,indr,rL);
88  %
89  % H) Second major segment (tapered). {sheet 1, c-d,4-5}:
90  indl = 3001;
91  indr = 4500;
92  rL = 5.40/2;
93  rR = 3.00/2;
94  rout(indl:indr) = geom_seg(indl,indr,rL,rR);
95  %
96  % I) Final section. {sheet 1, c-d,3-1}:
97  indl = 4501;
```

```
indr = 14500;
rL = 3.00/2;
rout(indl:indr) = geom_seg(indl,indr,rL);
%
indl = []; indr = []; rL = []; rR = [];
% _____
% Section (2)
% The approach here will be to record the inner diameters of the barrel in
% analogy with section 1.
% DEFINE: rin.
% USE: rout, author provided data as listed.
% .....
%
rin = zeros(size(rout)); % Initialize rin vector.
%
% A) Chamfer prior to rear threads. {sheet 2, b7}:
indl = 1;
indr = 18;
rL = (2.755 + 0.18)/2;
rR = 2.755/2;
rin(indl:indr) = geom_seg(indl,indr,rL,rR);

% B) Flat prior to main chamber. {sheet 2, b-c,7}:
indl = 19;
indr = 100;
rL = 2.755/2;
rin(indl:indr) = geom_seg(indl,indr,rL);
%
% C) Flat of main chamber. {sheet 2, b-c,7}:
indl = 101;
indr = 525;
rL = 2.81/2;
rin(indl:indr) = geom_seg(indl,indr,rL);
%
% D) Chamfer prior to forcing cone. {sheet 2, b-c,6}:
indl = 526;
indr = 536;
rL = 2.81/2;
rR = 2.750/2;
rin(indl:indr) = geom_seg(indl,indr,rL,rR);
%
% E) Flat prior to forcing cone. {sheet 2, b-c,6}:
indl = 537;
indr = 651;
rL = 2.750/2;
rin(indl:indr) = geom_seg(indl,indr,rL);
%
% F) Forcing cone. {sheet 2, b-c,6}:
indl = 652;
indr = 653;
rL = 2.496/2;
rR = 2.362/2;
rin(indl:indr) = geom_seg(indl,indr,rL,rR);
%
% G) Main barrel. {sheet 2, b-c,6-2}:
indl = 654;
indr = 14475;
rL = 2.362/2;
rin(indl:indr) = geom_seg(indl,indr,rL);
%
% H) The final chamfer. {sheet 2, b-c,2}:
indl = 14476;
indr = 14500;
rL = 2.362/2;
rR = 2.450/2;
rin(indl:indr) = geom_seg(indl,indr,rL,rR);
%
indl = []; indr = []; rL = []; rR = [];
% _____
% Section (3)
% Now to convert units to meters, and create a corresponding position
% vector in meters with indices corresponding to the rin & rout vectors.
% DEFINE: ns, spatial.
% USE: rin, rout.
% Alter: rin, rout.
% .....
%
rout = rout*(0.0254);% in(m/in)
rin = rin*(0.0254);% in(m/in)
ns = length(rout);% Number of geometric vector indices.
spatial = (1:ns)'*(1/100)*(0.0254);% (hundredths)(in/hund)(m/in)
% _____
% Section (4)
% Now compute the cross-sectional properties of the beam.
% Also compute the total mass to validate the geometry.
% DEFINE: lden, lEI, mass, masseng, deltax.
% USE: rin, rout, spatial, author provided data as listed.
% .....
%
deltax = mean(diff(spatial));% m (The disk thickness.)
density = 489*(0.4536)/(0.028317);% lbm/ft3(Kg/lbm)/(m3/ft3)
%       MARK's 9th, table page 6.44, Carbon steel (0.40% C).
%       Gun steel 4337m close to 4340 w/ some x-tra Vanadium
%       added for machinability. (4340 -> .38 - .43 % C)
lden = density*pi*(rout.^2 - rin.^2);% (Kg/m^3)(m^2)
%
mass = sum(lden)*deltax; % (Kg/m)(m)
masseng = mass*(1/0.4536);% Kg(1/(Kg/lbm))
%
E = (29.5*10^6)*6894.8; % lbf/in2*((N/m2)/(lbf/in2))
%  Private communication with Dr. Ron Gast, AMSTA-AR-CCB-DE.
%  (Shingley & Mitchell, 4th, table A-18 E's for steel from
%   29-30 Mpsi.) (Very heat treat dependent.)
%
lEI = E*(rout.^4 - rin.^4)*pi/4;
%
density = []; E = [];
% _____
% Section (5)
% Add in the external rigid mass. This mass is treated differently than
% simple non-beam mass (to be computed in section 6).It will be treated
% as a lumped inertia and directly coupled to the generalized
% coordinates of the first & or last node at a later time.
% For the case of a left external mass (Note: r would be negative):
%  x(ext) = x(1) + r*theta(1) -> dx(ext) = dx(1) + r*dtheta(1)
%  ddx(ext) = ddx(1) + r*ddtheta(1), ddtheta(ext) = ddtheta(1)
%  F(1) = mbr*ddx(ext), M(1) = (mext*r^2 + Jbr)*ddtheta(ext)
%  [m11 m12; m21 m22]*[ddx(1) ddtheta(1)]' = [F(1) M(1)]'
%
% DEFINE: ldenextl, ldenextr, Mextl, Mextr, spextl, spextr, massext.
% USE: deltax, author provided data as listed.
% .....
%
massext = 0;
%
% A) Left external mass: None.
Mextl = zeros(2,2);
spextl = [];
ldenextl = [];
massext = 0;
%
% B) Right external mass: Accelerator Unit from 3" prior to muzzle to 38"
%   (From Pat Vottis, 09 August 1995.)
%
% Initialize:
Mextr = zeros(2,2);
rextl = max(spatial) - 3*(0.0254);% in*(m/in)
rextr = max(spatial) + 38*(0.0254);% in*(m/in)
iextl = floor(rextl/deltax + ½);% Round to nearest increment of deltax.
iextr = floor(rextr/deltax + ½);
spextr = deltax*(iextl:iextr)';
ldenextr = zeros(size(spextr));
massext = 0;
%
% Rear Nut:
mext = 34*(0.4536);% lbm*(Kg/lbm)
rext = 0*(0.0254);% in*(m/in)
bext = 5*(0.0254);% in*(m/in)
hext = 5*(0.0254);% in*(m/in)
Jext = mext*(1/12)*(bext^2+hext^2);
Mextr = Mextr + [mext mext*rext; mext*rext (Jext + mext*rext^2)];
[y,indl] = min( abs(spextr - max(spatial) + 3*(0.0254)) );
[y,indr] = min( abs(spextr - max(spatial) - 2*(0.0254)) );
ldenextr(indl:indr) = ldenextr(indl:indr) + ...
        mext/(spextr(indr) - spextr(indl))*ones(size(ldenextr(indl:indr)));
%
% Casing:
mext = 32.4*(0.4536);% lbm*(Kg/lbm)
rext = 11*(0.0254);% in*(m/in)
bext = 22.2*(0.0254);% in*(m/in)
hext = 7*(0.0254);% in*(m/in)
Jext = mext*(1/12)*(bext^2+hext^2);
Mextr = Mextr + [mext mext*rext; mext*rext (Jext + mext*rext^2)];
[y,indl] = min( abs(spextr - max(spatial) + 0*(0.0254)) );
[y,indr] = min( abs(spextr - max(spatial) - 22*(0.0254)) );
ldenextr(indl:indr) = ldenextr(indl:indr) + ...
        mext/(spextr(indr) - spextr(indl))*ones(size(ldenextr(indl:indr)));
%
% G10's (4) & Liner:
mext = 12.4*(0.4536);% lbm*(Kg/lbm)
rext = 11*(0.0254);% in*(m/in)
bext = 19.4*(0.0254);% in*(m/in)
```

```
270 hext = 2.4*(0.0254);% in*(m/in)
271 Jext = mext*(1/12)*(bext^2+hext^2);
272 Mextr = Mextr + [mext mext*rext; mext*rext (Jext + mext*rext^2)];
273 [y,indl] = min( abs(spextr - max(spatial) - 1*(0.0254)) );
274 [y,indr] = min( abs(spextr - max(spatial) - 21*(0.0254)) );
275 ldenextr(indl:indr) = ldenextr(indl:indr) + ...
276         mext/(spextr(indr) - spextr(indl))*ones(size(ldenextr(indl:indr)));
277 %
278 % Coils (12) & Spacer:
279 mext = 9*(0.4536);% lbm*(Kg/lbm)
280 rext = 11*(0.0254);% in*(m/in)
281 bext = 12*(0.0254);% in*(m/in)
282 hext = 3*(0.0254);% in*(m/in)
283 Jext = mext*(1/12)*(bext^2+hext^2);
284 Mextr = Mextr + [mext mext*rext; mext*rext (Jext + mext*rext^2)];
285 [y,indl] = min( abs(spextr - max(spatial) - 5*(0.0254)) );
286 [y,indr] = min( abs(spextr - max(spatial) - 16*(0.0254)) );
287 ldenextr(indl:indr) = ldenextr(indl:indr) + ...
288         mext/(spextr(indr) - spextr(indl))*ones(size(ldenextr(indl:indr)));
289 %
290 % Forward Nut:
291 mext = 36*(0.4536);% lbm*(Kg/lbm)
292 rext = 22*(0.0254);% in*(m/in)
293 bext = 5*(0.0254);% in*(m/in)
294 hext = 5*(0.0254);% in*(m/in)
295 Jext = mext*(1/12)*(bext^2+hext^2);
296 Mextr = Mextr + [mext mext*rext; mext*rext (Jext + mext*rext^2)];
297 [y,indl] = min( abs(spextr - max(spatial) - 19*(0.0254)) );
298 [y,indr] = min( abs(spextr - max(spatial) - 24*(0.0254)) );
299 ldenextr(indl:indr) = ldenextr(indl:indr) + ...
300         mext/(spextr(indr) - spextr(indl))*ones(size(ldenextr(indl:indr)));
301 %
302 % Muzzle Extension:
303 mext = 18.2*(0.4536);% lbm*(Kg/lbm)
304 rext = 30.5*(0.0254);% in*(m/in)
305 bext = 15*(0.0254);% in*(m/in)
306 hext = 4*(0.0254);% in*(m/in)
307 Jext = mext*(1/12)*(bext^2+hext^2);
308 Mextr = Mextr + [mext mext*rext; mext*rext (Jext + mext*rext^2)];
309 [y,indl] = min( abs(spextr - max(spatial) - 23*(0.0254)) );
310 [y,indr] = min( abs(spextr - max(spatial) - 38*(0.0254)) );
311 ldenextr(indl:indr) = ldenextr(indl:indr) + ...
312         mext/(spextr(indr) - spextr(indl))*ones(size(ldenextr(indl:indr)));
313 % ____________________________________________________________________
314 % Section (6)
315 % Add in non-beam masses. (Since this data is not often directly on
316 % drawings, the nearest indices of spatial are found from positions
317 % expressed in general units of measure, rather than indices directly.)
318 % Also compute the total non-beam mass to validate the geometry.
319 % This section uses geom_nbseg.m.
320 % DEFINE: lnbden, massnb.
321 % USE: deltax, ns, spatial, author provided data as listed.
322 % .....................................................................
323 %
324 lnbden = zeros(ns,1); % Initialize lnbden vector.
325 %
326 % A) The Breech, mb = 50#, located 0.75" RFT spread over 1.5".
327 %
328 massnb = 50*(0.4536);% lbm*(Kg/lbm) Non beam mass.
329 posl = (0.01)*(0.0254);% in(m/in) Left most position of distribution.
330 posr = (1.50)*(0.0254);% in(m/in) Right most position of distribution.
331 %
332 % Identify indices of lennb and effective linear density of mnb.
333 [indl,indr,lnbdenseg] = geom_nbseg(posl,posr,massnb,spatial);
334 %
335 % The non-beam mass is now added into the lnbden vector:
336 lnbden(indl:indr) = lnbden(indl:indr) + lnbdenseg;
337 %
338 % B) No additional non-beam masses. Sum up total mass.
339 %
340 massnb = (sum(lnbden)+sum(ldenextl)+sum(ldenextr))*deltax;
341 %
342 %
343 indl = []; indr = []; lnbdenseg = []; posl = []; posr = [];
344 % ____________________________________________________________________
345 % Section (7)
346 % Define the geometry matrix. (Used for plotting.)
347 % DEFINE: gm.
348 % USE: rin, rout.
349 % .....................................................................
350 %
351 gm = [rin rout];
352 % ____________________________________________________________________
353 % Section (8)
354 % If flag is set, plot the property vectors and geometric matrix for
355 % visual inspection in the current figure window. (Note: in order to
356 % preserve the visual inspection, the non-beam masses are plotted
357 % within the confines of the geometry matrix. Otherwise the visual
358 % validation of linear beam density could be obscured.)
359 % USE: deltax, lEI, gm, lden, lnbden, spatial, tlabel.
360 % .....................................................................
361 %
362 if nargin == 1
363  clg
364  %
365  % A) Determine if tlabel is a valid title string or apply default:
366  %
367  if isstr(tlabel); % Checks to see if tlabel is a string.
368   tlabel(1) = upper(tlabel(1)); % Imposes uppercase on first letter.
369  else
370   tlabel = ['Hybrid 60mm'];% Default is used if no valid tlabel given.
371  end
372  %
373  % B) Combine external and non beam masses into one vector:
374  %
375  nbxlden = lnbden;% Initialize the combined external and non beam
376  %    mass vector that is within the confines of the beam geometry.
377  %
378  lobindx = find(spextl < deltax);% Left of beam index.
379  lobspat = spextl(lobindx);% Left of beam spatial vector.
380  loblden = ldenextl(lobindx);% Left of beam linear density.
381  coindx = find(spextl >= deltax);% Index of ldenextl within beam.
382  spatindx = (1:length(coindx))';% Indices of ldenextl in terms of
383  %                    spatial, within beam.
384  % Now combine the left external and non beam linear density that
385  % coexists within the confines of the beam geometry.
386  nbxlden(spatindx) = nbxlden(spatindx) + ldenextl(coindx);
387  %
388  robindx = find(spextr > max(spatial));% Right of beam index.
389  robspat = spextr(robindx);% Right of beam spatial vector.
390  roblden = ldenextr(robindx);% Right of beam linear density.
391  coindx = find(spextr <= max(spatial));% Index of ldenextr within beam.
392  spatindx = ((ns-length(coindx)+1):ns)';% Indices of ldenextr in terms of
393  %                    spatial, within beam.
394  % Now combine the right external and non beam linear density that
395  % coexists within the confines of the beam geometry.
396  nbxlden(spatindx) = nbxlden(spatindx) + ldenextr(coindx);
397  %
398  % C) Determine plotting scale to force external and non beam densities
399  %   to be plotted within the confines of the beam geometry.
400  %
401  nbi = find(nbxlden);% Find external & non beam density index locus.
402  psf = (0.50)*min( min(gm(nbi,:))'./nbxlden(nbi) );% Plot scale factor.
403  %
404  % D) Combine external and non beam mass totals:
405  %
406  nbmass = (sum(lnbden)+sum(ldenextl)+sum(ldenextr))*deltax;
407  nbmasseng = nbmass*(1/0.4536); % (Kg)(1/(Kg/lbm))
408  %
409  % E) Plot results:
410  %
411  subplot(311), plot(spatial,gm,'k'); hold on
412   plot(spatial,-gm,'k');
413   stem(spatial(nbi),psf*nbxlden(nbi));
414   stem(spatial(nbi),-psf*nbxlden(nbi));
415   stem(lobspat,psf*loblden); stem(lobspat,-psf*loblden);
416   stem(robspat,psf*roblden); stem(robspat,-psf*roblden);
417   hold off
418   tstring = [tlabel ' Profile & Non beam Masses Versus Length '...
419         'in Meters, (Total Non beam Mass of ' ...
420          num2str(nbmass) ' Kg, or ' num2str(nbmasseng) ' lbm)'];
421   title(tstring)
422   ylabel('m')
423   axis([max(spatial)*[-0.1 1.3], max(rout)*[-1.2 1.2]]); grid;
424   %
425   % The following axis manipulation scales the subplot for better printing.
426   pos = get(gca,'position'); % This is in normalized coordinates
427   pos(4)=pos(4)*.85;      % Shrink the height by a factor of .85
428   pos(2)=pos(2) + pos(4)*.15;% Raise the subplot by the saved height.
429   set(gca,'position',pos);
430   %
431  subplot(312), plot(spatial,lden,'k');
432   tstring = ['Linear Density from Profile, (Total Mass of Beam ' ...
433         num2str(mass) ' Kg, or ' num2str(masseng) ' lbm)'];
434   title(tstring)
435   ylabel('Kg/m')
436   axis([max(spatial)*[-0.1 1.3], max(lden)*[0 1.2]]); grid;
437   %
438   % The following axis manipulation scales the subplot for better printing.
439   pos = get(gca,'position'); % This is in normalized coordinates
440   pos(4)=pos(4)*.85;      % Shrink the height by a factor of .85
441   pos(2)=pos(2) + pos(4)*.15;% Raise the subplot by the saved height.
```

```
442    set(gca,'position',pos);
443    %
444   subplot(313), plot(spatial,lEI,'k');
445    ylabel('N-m^2')
446    xlabel('Axial Position (m)')
447    title('EI from Profile')
448    axis([max(spatial)*[-0.1 1.3], max(lEI)*[0 1.2]]); grid;
449    %
450    % The following axis manipulation scales the subplot for better printing.
451    pos = get(gca,'position'); % This is in normalized coordinates
452    pos(4)=pos(4)*.85;        % Shrink the height by a factor of .85
453    pos(2)=pos(2) + pos(4)*.15;% Raise the subplot by the saved height.
454    set(gca,'position',pos);
455    %
456 end
457 %
458 coindx = []; lobindx = []; loblden = []; lobspat = []; nbi = [];
459 nbmass = []; nbmasseng = [];  nbx = []; den = []; pos = []; psf = [];
460 robindx = []; roblden = []; robspat = []; spatindx = []; tstring = [];
461 % ____________________________________________________________________
462 % Completed:
463 % OUTPUT: gm, lden, lEI, lnbden, Mextl, Mextr, spatial.
464 % ....................................................................
465 %
```

<hybrid60.m>

```
1  % hybrid60.m -> A working script M-file to execute analysis of the hybrid
2  % 60mm gun using the finite element formulation and dynamics analysis
3  % functions defined within the subdirectory, beam_fem/.
4  % o   Created 02 January - 09 January 1996 by Eric Kathe.
5  %   Benet Labs, Watervliet Arsenal, NY  12189-4050 <ekathe@pica.army.mil>
6  % ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
7
8  % ____________________________________________________________________
9  % Section (1)
10 % Set the plot and print flags on or off, run the geometry
11 % m-file, and create a plot label.
12 % DEFINE: tlabel, spatial, lden, lEI, lnbden, Mextl, Mextr, gm.
13 % ....................................................................
14 %
15 % Set plot and print flags to 1 to enable, zero to disable.
16 plot_on = 1;
17 print_on = 0;
18 %
19 if print_on == 1
20   plot_on = 1;  % Clearly, to print, the plot flag must be enabled.
21 end
22 %
23 tlabel = 'Hybrid 60mm';% tlabel -> Enables plotting. If valid, the
24 %  string of tlabel is incorporated into the plot to describe the beam.
25 %
26 if plot_on == 1
27   figure(1)
28   clg
29   set(gcf,'PaperOrientation','portrait');  % This series of commands configures
30   set(gcf,'PaperUnits','inches');          % the plot window to effectively
31   set(gcf,'PaperPosition',[1 1 6.5 4.5]);  % be incorporated into a report.
32   set(gcf,'Units','inches','position',get(gcf,'PaperPosition'));
33   set(gcf,'DefaultAxesFontSize',9);
34   [spatial, lden, lEI, lnbden, Mextl, Mextr, gm] = geomf_hybrid(tlabel);
35 else
36   [spatial, lden, lEI, lnbden, Mextl, Mextr, gm] = geomf_hybrid;
37 end
38 %   spatial -> Axial position vector.
39 %   lden -> Beam linear density vector.
40 %   lEI -> Similar to lden except for linear EI cross-section properties.
41 %   lnbden -> The inertia of non-beam masses attached to the beam.
42 %   Mextl, Mextr -> 2x2 sub matrices of left and right extreme rigid body
43 %                   inertia.
44 %   gm -> The columns of this matrix record the inner and outer radii.
45 if print_on == 1
46   print -deps fig15.ps; % Print the file as an encapsulated Post-Script file.
47 end
48 %
49 % ____________________________________________________________________
50 % Section (2)
51 % Define barrel constraint locations, number of elements, and create
52 % element mesh vector.
53 % DEFINE: snlv, ncord.
54 % USE: spatial, lden, lEI, lnbden, snlv, tlabel, Author supplied
55 %      data, Author supplied default value.
56 % ....................................................................
57 %
58 hang1 = 15*(0.0254);% in(m/in) Location of hanger 1, near breach.
59 hang2 = 100*(0.0254);% in(m/in) Location of hanger 2, near muzzle.
60 snlv = [hang1; hang2];% (m) Imposed node location vector.
61 %
62 nel = 74;% Number of elements.
63 %
64 [ncord] = fem_mesh(spatial,lden,lEI,zeros(size(lnbden)),snlv,nel);
65 % ____________________________________________________________________
66 % Section (3)
67 % Generate the beam-only mass and stiffness matrices via FEM.
68 % DEFINE: Mfem, Kfem.
69 % USE: spatial, lden, lEI, ncord, tlabel
70 % ....................................................................
71 %
72 [Mfem,Kfem] = fem_form(spatial,lden,lEI,zeros(size(lden)),ncord);
73 %  Mfem & Kfem are out-put mass and stiffness matrices.  The non-beam
74 %  linear density vector is set to zero, to consider just the barrel.
75 % ____________________________________________________________________
76 % Section (4)
77 % Generate the free-free system constraints (zero).
78 % DEFINE: constraintm.
79 % USE: author supplied data.
80 % ....................................................................
81 %
82 kcnst = [0; 0];% Set constraint springs to zero column vector.
83 %
84 cdcnst = 0*kcnst; % Zero damping column vector.
85 %
86 [y, k1] = min(abs(snlv(1) - spatial(ncord)));% Identification of
87 [y, k2] = min(abs(snlv(2) - spatial(ncord)));% constraint node number.
88 %
89 gcindk = 2*[k1;k2] - 1;% Identification of lateral constraint generalized
90 %                      coordinate number.
91 %
92 constraintm = [gcindk kcnst cdcnst];
93 %
94 kcnst = []; cdcnst = []; y = []; k1 = []; k2 = []; gcindk = [];
95 % ____________________________________________________________________
96 % Section (5)
97 % Compute system matrices without any external nor non-beam masses.
98 % Also generate the Rayleigh damping matrix.
99 % DEFINE: M, K, Cd, n2.
100 % USE: author supplied data.
101 % ....................................................................
102 %
103 [M,K,Cd] = fem_lump(Mfem,Kfem,zeros(2,2),zeros(2,2),0,0,constraintm);
104 %   M, K, Cd -> Mass, stiffness, and zero damping matrices of
105 %         generalized coordinates that include the constraint and
106 %         externally coupled dynamics.
107 n2 = size(M,1);% The number of generalized coordinates.
108 %
109 % ____________________________________________________________________
110 % Section (6)
111 % Generate the second order, undamped eigen-modes.
112 % DEFINE: phi, fvn, rlab.
113 % USE: author supplied data.
114 % ....................................................................
115 %
116 [phi,fvn,rlab] = eigen_2o(M,K);% Compute, normalize, sort, and identify
117 %                              the modes of vibration.
118 % ____________________________________________________________________
119 % Section (7)
120 % Import Dr. Gast's Mode shapes & compare (if available).
121 % DEFINE: phi, fvn, rlab.
122 % USE: author supplied data.
123 % ....................................................................
124 %
125 ml_phi = zeros(length(spatial),6); % Initialize matrix for MATLAB modes.
126 rg_phi = zeros(length(spatial),6); % Initialize zero matrix if no modes.
127 rg_nx = spatial/max(spatial);% Initialize axial position if no modes.
128 %
129 modes_avail = 1;% Flag if Dr. Gast's modes are available. If not, set to 0.
130 %
131 for I = 1:6
132   if modes_avail == 1
133     eval(['load /usr/people/usrrt/ekathe/wpfiles/gast_hyb/mode' ...
134           int2str(I) '.asc;']);
135     eval(['mat = mode' int2str(I) ';']);
136     eval(['clear mode' int2str(I) ';']);
137     if I == 1
138       rg_phi = zeros(size(mat,1),6); % Initialize matrix for modes.
139       rg_nx = mat(:,1);% Respective axial position vector.
140     end
141     rg_phi(:,I) = mat(:,2)/max(abs(mat(:,2)));
142     rg_phi(:,I) = rg_phi(:,I)/max(abs(rg_phi(:,I)));
143     clear mat;
144   end
145   %
146   ml_phi(:,I) = mode_shape(ncord,spatial,-phi(:,(I+2)));
```

```
ml_phi(:,I) = ml_phi(:,I)/max(abs(ml_phi(:,I)));% Unit normalize.
end
%
ml_nx = spatial/max(spatial);% Unit normalize axial position vector.
frg = [26.4 83.1 187.4 329.5 478.2 656.4]'; % Dr. Gast's frequencies.
fml = fvn(3:8);% FEM frequencies, minus the first two rigid body modes.
%
if plot_on == 1
 figure(2)
 clg
 set(gcf,'PaperOrientation','portrait');  % This series of commands configures
 set(gcf,'PaperUnits','inches');          % the plot window to effectively
 set(gcf,'PaperPosition',[1 1 6.5 4.5]);  % be incorporated into a report.
 set(gcf,'Units','inches','position',get(gcf,'PaperPosition'));
 set(gcf,'DefaultAxesFontSize',10);
 n = 0;
 for j = 1:3
  for k = 0:3:3
   I = j+k;
   n = n+1;
   subplot(3,2,n)
   plot(ml_nx,ml_phi(:,I),'k:',rg_nx,rg_phi(:,I),'k');
   mlabel = rlab(I+2,fmd(abs(rlab(I+2,:)) ~= 32));
   mlabel = [mlabel ', :FEM ' num2str(fml(I)) 'Hz:'];
   if modes_avail == 1
    mlabel = [mlabel ', -USM ' num2str(frg(I)) ' Hz-'];
   end
   title(mlabel)
   axis([-0.05 1.05 -1.05 1.05])
   if n > 4
    xlabel('Normalized Axial Position')
   end
   pos = get(gca,'position'); % This is in normalized coordinates
   pos(4)=pos(4)*.85;        % Shrink the height by a factor of .85
   pos(2)=pos(2) + pos(4)*.15;% Raise the subplot by the saved height.
   set(gca,'position',pos);
   if modes_avail == 1
    if I < 4
     legend('FEM','USM');
    end
   end
  end
 end
 if modes_avail == 1
  subtitle('60mm Barrel Eigenvectors and Frequencies (:FEM:, -USM-)')
 else
  subtitle('60mm Barrel Eigenvectors and Frequencies')
 end
end
%
if print_on == 1
 print -deps fig16.ps; % Print the file as an encapsulated Post-Script file.
end
%
% _____________________________________________________________________
% Section (8)
% Generate the as hung system constraints.
% DEFINE: constraintm.
% USE: author supplied data.
% .....................................................................
%
kcnst = [1; 1]*2*10^5;% Set constraint springs to 200,000 N/m.
%
cdcnst = 0*kcnst; % Zero damping column vector.
%
[y, k1] = min(abs(snlv(1) - spatial(ncord)));% Identification of
[y, k2] = min(abs(snlv(2) - spatial(ncord)));% constraint node number.
%
gcindk = 2*[k1;k2] - 1;% Identification of lateral constraint generalized
%                      coordinate number.
%
constraintm = [gcindk kcnst cdcnst];
%
kcnst = []; cdcnst = []; y = []; k1 = []; k2 = []; gcindk = [];
% _____________________________________________________________________
% Section (9)
% Compute system matrices with all external and non-beam masses.
% DEFINE: M, Mfem, K, Kfem, Cd, n2.
% USE: author supplied data.
% .....................................................................
%
[Mfem,Kfem] = fem_form(spatial,lden,EI,lnbden,ncord);
%
[M,K,Cd] = fem_lump(Mfem,Kfem,Mextl,Mextr,0,0,constraintm);
%
n2 = size(M,1);% The number of generalized coordinates.
%
% _____________________________________________________________________
% Section (10)
% Force due to gravity.
% USE: A, B, C, D.
% .....................................................................
g = -9.8067; % Earth Surface Gravity (m/(s^2)).
weight = g*(lden + lnbden)*diff(spatial(1:2));
%
% [F] = fem_force(spatial,weight,zeros(size(lden)),ncord,tlabel);% Prints F.
[F] = fem_force(spatial,weight,zeros(size(lden)),ncord);
%
% Now incorporate the two 2x2 rigid body mass matrices after checking to
% avoid divide by zero:
%
if abs(Mextl(1,1)) > 0
 F(1) = F(1) + g*Mextl(1,1); % g*Kg.
 F(2) = F(2) + g*(Mextl(1,2)^2/Mextl(1,1)); % g*( (m*Kg)^2/Kg).
end
%
if abs(Mextr(1,1)) > 0
 F(n2-1) = F(n2-1) + g*Mextr(1,1);
 F(n2) = F(n2) + g*(Mextr(1,2)^2/Mextr(1,1)); % g*( (m*Kg)^2/Kg).
end
% _____________________________________________________________________
% Section (11)
% Second order inverted stiffness approximation to static gravity loading.
% USE: K and F.
% .....................................................................
xinvk = K\F;% Ie, since x ddot -> zeros, K*x = F -> x = inv(K)*F.
% _____________________________________________________________________
% Section (12)
% Plot the static gravity deflection.
% USE: xinvk, gm, spatial, M.
% .....................................................................
sm = 2*floor(size(M,1)/2);
xtemp = xinvk(1:2:sm);% Use lateral displacements of nodes for scaling.
scale = max(abs(xtemp))/(8*max(max(gm)));
av = [ (min(spatial)- 0.1*max(spatial)) 1.1*max(spatial) ...
    [-15 5]/1000];
%
if plot_on == 1
 figure(3)
 clg
 set(gcf,'PaperOrientation','portrait');  % This series of commands configures
 set(gcf,'PaperUnits','inches');          % the plot window to effectively
 set(gcf,'PaperPosition',[1.75 3.75 4 2.5]);  % be incorporated into a report.
 set(gcf,'Units','inches','position',get(gcf,'PaperPosition'));
 set(gcf,'DefaultAxesFontSize',10);
 %
 beam_plot(ncord,spatial,xinvk,gm*scale,constraintm(:,1));
 % av = axis;
 % av(1) = -av(2)/10;
 axis(av)
 grid
 title(['      ' tlabel ' Static Gravity Deflection'])
 xlabel('Axial Position (m)')
 ylabel('Lateral Deflection (m)')
 pos = get(gca,'position');          % This is in normalized coordinates
 pos(2)=pos(2)+(1-0.85)*pos(4)/2; % Raise the subplot & keep it centered.
 pos(4)=0.85*pos(4);                % Shrink the height by a factor of .85
 pos(1)=pos(1)+(1-0.85)*pos(4)/2; % Shift to the right.
 set(gca,'position',pos);
end
%
if print_on == 1
 print -deps fig17.ps; % Print the file as an encapsulated Post-Script file.
end
% _____________________________________________________________________
% Completed:
% .....................................................................
```

<mode_shape.m>

```
function [phi] = mode_shape(ncord,spatial,y)
% mode_shape.m
% [] = mode_shape(ncord,spatial,y);
% o  This m-file generates a mode shape of a beam in its spatial domain.
% o  This file uses:
%      fem_node_check.m in section 1.
%        geom_check.m
%      fem_interp.m in section 2.
% o  External generalized coordinates may be include via extgc.
% o  Specific input/output variable definitions are:
%  ncord -> The vector of indices such that spatial(ncord) is the position
%    of each pair of beam generalized coordinates. (Also known as nodes.)
%  spatial -> Axial position vector, currently limited to equally spaced
```

```
%     position vectors with position data of every dx*n point such that
%     n = 1:length(spatial) and length(spatial)*dx equals max(spatial).
%  y -> Output vector. Contains the magnitudes of the displacements and
%     rotations of the generalized coordinates. It is a vector of the form:
%     {x(1) theta(1) x(2)...x(n2) theta(n2) extgc(1) extgc(2)...extgc(nxgc)}'.
%  phi -> mode shape vector that corresponds one-to-one with spatial.
% o  Created 3 January 1996 by Eric Kathe.
%  Benet Labs, Watervliet Arsenal, NY  12189-4050 <ekathe@pica.army.mil>
% ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
% _____________________________________________________________________________
% Section (1)
% A few checks to be sure input variables are the right size, et cetera.
% This section uses geom_check.m to validate the input geometry vectors.
% Also define the number of indices of spatial, ns, the number of generalized
% coordinates ngc, et cetera.
% DEFINE: nel, ngc, nn, ns, nxgc.
% USE: nargin, ncord, spatial, y,
%    author provided default value.
% POSSIBLY ALTER: ncord, spatial, y.
% .............................................................................
% (A) Check spatial & ncord & define their length's, ns.
%
[ncord,spatial] = fem_node_check(ncord,spatial);
ns = length(spatial); % Number of indices of spatial.
nn = length(ncord); % Number of nodes of ncord.
nel = nn - 1;% Number of elements of the beam.
%
%
% (B) Impose column structure on y, and check constancy of number of
% generalized coordinates.
%
y = y(:);
ngc = length(y);
if ngc ~= (2*nn)
  if ngc < 2*nn
    warning = ['More generalized coordinates than the number of outputs.'...
          'No corrective action has been taken.']
  elseif nargin == 5
    zgext = ones((ngc - 2*nn),1);
    zgext(1:nxgc) = extgc;
    extgc = zgext(1:(ngc - 2*nn));
    warning = ['Mismatched number of generalized & or external '...
          'couplings.  extgc padded or truncated to fit.  '...
          'Further errors are very likely.']
  else
    warning = ['Fewer generalized coordinates than the number of outputs.'...
          'No corrective action has been taken.']
  end
end
%
default = []; warning = [];
% _____________________________________________________________________________
% Section (2)
% Compute the beam deflection vectors for each finite element using
% fem_interp.m, and combine into one beam deflection vector.
% DEFINE: deflectv.
% USE: ncord, nn, ns, spatial, y,
% .............................................................................
%
% (A) Initialize deflection vector.
%
deflectv = zeros(size(spatial));
%
% (B) For the first element of the beam.
%
lengvseg = spatial(ncord(1):ncord(2)); % Length of finite element.
[zphi, zddphi] = fem_interp(lengvseg);% Zphi contains the shape functions.
%
xv = y(1:4);% The deflection of element one is a function of the
%          state of both adjacent nodes. In this case
%          {x1, theta1, x2, theta2}.
%
deflectv(ncord(1):ncord(2)) = zphi*xv;
%
% © For remaining elements:
%
for I=2:nel;
  %
  lengvseg = spatial((ncord(I)+1):ncord(I+1)) - spatial(ncord(I));
  [zphi, zddphi] = fem_interp(lengvseg);
  indL = 2*I - 1;
  indR = indL + 3;
  deflectv((ncord(I)+1):ncord(I+1)) = zphi*y(indL:indR);
end
%
phi = deflectv;
%
% _____________________________________________________________________________
I = []; indL = []; indR = []; xv = []; lengvseg = []; zphi = [];
zddphi = [];
% _____________________________________________________________________________
% Completed:
% OUTPUT: None.
% .............................................................................
```

<rank_ki.m>

```
% rank_ki.m
% rank_ki.m -> A script M-file that utilizes the symbolic toolbox to
%          demonstrate the deficient rank of the elemental
%          stiffness matrix.
% o  Created 06 - 07 November 1995 by Eric Kathe.
%  Benet Labs, Watervliet Arsenal, NY  12189-4050 <ekathe@pica.army.mil>
% ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
% _____________________________________________________________________________
% Section (1)
% Enter the Hermite-cubic polynomials.  (Junkins & Kim:  (4.101)
% DEFINE: phi1, phi2, phi3, phi4.
% .............................................................................
%
phi1 = '1 - 3*x^2/(h^2) + 2*x^3/(h^3)';
phi2 = 'x - 2*h*x^2/(h^2) + h*x^3/(h^3)';
phi3 = '3*x^2/(h^2) - 2*x^3/(h^3)';
phi4 = '-h*x^2/(h^2) + h*x^3/(h^3)';
%
% _____________________________________________________________________________
% Section (2)
% Compute the second spatial derivative pairs of the Hermite-cubic polynomials
% as used in the integration definition of the elemental stiffness values.
% Also print them.
% (Junkins & Kim:  (4.102)):
% USE: phi1, phi2, phi3, phi4.
% .............................................................................
% DISABLED:  Used to display each multiplied pair of second spatial
%          differentiated interpolation functions.
% for I = 1:4
%  for j = 1:i
%    [I j]
%    eval(['pretty( expand(symmul(diff(phi' int2str(I) ',2),diff(phi' ...
%                        int2str(j) ',2))))'])
%  end
% end
%
I = []; j = [];
% _____________________________________________________________________________
% Section (3)
% Compute the first three independent pairs of section (2).
% DEFINE: a, b, g.
% .............................................................................
%
a = expand(symmul(diff(phi1,2),diff(phi1,2)));
b = expand(symmul(diff(phi2,2),diff(phi1,2)));
g = expand(symmul(diff(phi2,2),diff(phi2,2)));
% _____________________________________________________________________________
% Section (4)
% Validate the following matrix simplification, smat, element by element.
%
% Smat is to be a simplification of the multiplied pairs of second
% spatial derivatives of interpolation functions that are subsequently
% multiplied by the axial distribution of elemental stiffness, and then
% integrated from zero to the element length, h, to form the symmetric
% elemental stiffness matrix. (See Junkins & Kim:  (4.102))
%
% DEFINE: check11, check21, check22, check31, check32, check33,
%       check41, check42, check43, check44, smat.
% USE: phi1, phi2, phi3, phi4, a, b, g.
% .............................................................................
%
smat = str2mat('[  a     -     -     Sym   ]',...
          '[  b     g     -     -     ]',...
          '[ -a    -b     a     -     ]',...
          '[ (a*h-b)  (b*h-g)  -(a*h-b)  (a*h^2-2*b*h+g) ]');
%
% Subtract a from stiffness interpolation pair
% of matrix element (1,1):
check11 = simple( symsub( symmul(diff(phi1,2),diff(phi1,2)) , a) );
% Subtract b from stiffness interpolation pair
% of matrix element (2,1):
check21 = simple( symsub( symmul(diff(phi2,2),diff(phi1,2)) , b) );
% Subtract g from stiffness interpolation pair
% of matrix element (2,2):
check22 = simple( symsub( symmul(diff(phi2,2),diff(phi2,2)) , g) );
% Subtract -a from stiffness interpolation pair
```

```
% of matrix element (3,1):
check31 = simple( symadd( symmul(diff(phi3,2),diff(phi1,2)) , a) );
% Subtract -b from stiffness interpolation pair
% of matrix element (3,2):
check32 = simple( symadd( symmul(diff(phi3,2),diff(phi2,2)) , b) );
% Subtract a from stiffness interpolation pair
% of matrix element (3,3):
check33 = simple( symsub( symmul(diff(phi3,2),diff(phi3,2)) , a) );
% Subtract (a*h-b) from (4,1):
check41 = simple( symsub( symmul(diff(phi4,2),diff(phi1,2)) , ...
                          symsub( symmul(a,'h') ,b) ) );
% Subtract (b*h-g) from stiffness interpolation pair
% of matrix element (4,2):
check42 = simple( symsub( symmul(diff(phi4,2),diff(phi2,2)) , ...
                          symsub( symmul(b,'h') ,g) ) );
% Subtract -(a*h-b) from stiffness interpolation pair
% of matrix element (4,3):
check43 = simple( symadd( symmul(diff(phi4,2),diff(phi3,2)) , ...
                          symsub( symmul(a,'h') ,b) ) );
% Subtract -(a*h^2-2*b*h+g) from stiffness interpolation pair
% of matrix element (4,4):
check44 = simple( symsub( symmul(diff(phi4,2),diff(phi4,2)) , ...
          symadd(symsub(symmul(a,'h^2'),symmul(b,'2*h')),g) ) );
%
% _____________________________________________________________________
% Section (5)
% Examine results for errors.
% USE:   check11, check21, check22, check31, check32, check33,
%        check41, check42, check43, check44.
% ....................................................................
%
eflag = 0;
errormat = zeros(4,4);
for I = 1:4
  for j = 1:i
   eval(['check = check' int2str(I) int2str(j) ';'])
   if check ~= '0'
    warning = ['matrix element (' int2str(I) ',' int2str(j) ...
                                  ') results in error.']
    eflag = eflag + 1
    errormat(i,j) = 1;
    errormat(j,i) = 1;
   end
  end
end
if eflag == 0
  message = ' No errors found. The matrix, smat, was validated:'
  smat
else
  warning = [ int2str(eflag) ' errors found.  The matrix is not validated.']
  smat
  message = 'Error locations indicated by a one in the following matrix.'
  errormat
end
%
I = []; j = []; eflag = []; message = []; warning = [];
% _____________________________________________________________________
% Completed:
% ....................................................................
%
```

<ubeameig.m>

```
function [ccosh] = ubeameig(f)
% ubeameig.m
% [ccosh] = ubeameig(f);
% o This function numerically computes the uniform beam eigenvalue
%   equation.  The zero crossings are solutions of it.
% o Created January 1996 by Eric Kathe.
%   Benet Labs, Watervliet Arsenal, NY  12189-4050 <ekathe@pica.army.mil>
% ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

ccosh = 1 - cos(f).*cosh(f);
```

<uniform.m>

```
% uniform.m -> A working script M-file to execute analyses of
% uniform beams.
% o   Created 10 January 1996 by Eric Kathe.
%   Benet Labs, Watervliet Arsenal, NY  12189-4050 <ekathe@pica.army.mil>
% ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

% _____________________________________________________________________
% Section (1)
% Compute approximations of the uniform beam geometry for finite element
% analysis, and numerical approximation of analytic mode-shapes.
% DEFINE: Xn, L, rho, E, I, spatial, lden, lEI, lnbden, gm.
% ....................................................................
%
% Set plot and print flags to 1 to enable, zero to disable.
plot_on = 1;
print_on = 0;
%
if print_on == 1
  plot_on = 1;  % Clearly, to print, the plot flag must be enabled.
end
%
% Define a normalize axial position vector from 0 < x <= 1:
%
Xn = (1:1001)'/1001;
%
% Set uniform beam properties to unity.
%
L = 1;
rho = 1;
E = 1;
I = 1;
spatial = Xn;
lden = rho*ones(size(spatial));
lEI = E*I*ones(size(spatial));
lnbden = zeros(size(lden));
gm = ones(size(spatial));
% _____________________________________________________________________
% Section (2)
% Compute Analytic Eigenvectors and bending frequencies.  Symbolic commands
% to derive solution are shown.
% DEFINE: fub, ub_phi.
% USE: nx, L, rho, E, I.
% ....................................................................
%
% (A) Define and display initial calculations and application of boundary
%    conditions.
%
% Define assumed form of the mode shape function:
%
shape = ['C1*(cos(k*x) + cosh(k*x)) + C2*(cos(k*x) - cosh(k*x)) ' ...
        '+ C3*(sin(k*x) + sinh(k*x)) + C4*(sin(k*x) - sinh(k*x))'];
%
% Display free-free boundary conditions, this solution shows that C2 and
% C4 must be zero:
%
% simple(subs(diff(shape,2),'0'))
%  ans = -2*C2*k^2
% simple(subs(diff(shape,3),'0'))
%  ans = -2*C4*k^3
% simple(subs(diff(shape,2),'L'))
%  ans = (C1*(-cos(k*L)+cosh(k*L))+C2*(-cos(k*L)-cosh(k*L))+...
%        C3*(-sin(k*L)+sinh(k*L))+C4*(-sin(k*L)-sinh(k*L)))*k^2
% simple(subs(diff(shape,3),'L'))
%  ans = (C1*(sin(k*L)+sinh(k*L))+C2*(sin(k*L)-sinh(k*L))+...
%        C3*(-cos(k*L)+cosh(k*L))+C4*(-cos(k*L)-cosh(k*L)))*k^3
%
% Define mode shape with C2 and C4 set to zero:
%
shapet = subs(subs(shape,'0','C2'),'0','C4');
%
% simple(subs(diff(shapet,2),'L'))
%  ans = (C1*(-cos(k*L)+cosh(k*L))+C3*(-sin(k*L)+sinh(k*L)))*k^2
% simple(subs(diff(shapet,3),'L'))
%  ans = (C1*(sin(k*L)+sinh(k*L))+C3*(-cos(k*L)+cosh(k*L)))*k^3
%
% Define 2x2 matrix that when multiplied by [C1 C3]' results in [0 0]'
% when the remaining two boundary conditions are satisfied.  This matrix
% is assembled by cut&paste from above commented-out simple commands:
%
dmat = sym(['-cos(k*L)+cosh(k*L),-sin(k*L)+sinh(k*L);' ...
          'sin(k*L)+sinh(k*L),-cos(k*L)+cosh(k*L)']); % Symbolic form.
%
% Evaluate and simplify the eigen-determinate for values of (k*L) that
% satisfy the remaining two boundary conditions excluding the trivial
% solution. ( C1 = 0 & C2 = 0 is trivial.)
%
% >> pretty(simple(determ(dmat)))
%
%                    2 - 2 cos(k L) cosh(k L)
%
% For MATLAB to find the roots of this eigenvalue problem, the determinate
% equation must be implemented in a new function file, <ubeameig.m>, as
% shown below, then the function <fzero> may be applied to solve for the
% roots closest to an initial guess that must be provided:
% ^^^^------
% function [ccosh] = ubeameig(f)
% %
% ccosh = 1 - cos(f).*cosh(f);
% ^^^^------
```

```
%
% (B) Numerically identify the eigenvalues:
%
% For the uniform beam being analyzed, w = k^2*(E*I/rho)^(½).
% Also, w = f*(2*pi), (radians/time) = (cycles/time)*(2*pi).
%
% Define high resolution search vector that exceeds the highest
% expected root by 20%.
%
kig = ((1:1000)'/1000)*25;% 25 ~ sqrt(w 6-mode rad/sec)*1.2
rts = zeros(size(kig)); % Initialize roots vector.
%
% Find the roots of the eigenvalue equation using the search vector
% values as initial guesses and compute cyclic frequencies:
%
for I = 1:length(kig)
 rts(I) = fzero('ubeameig',kig(I));
end
% plot(kig,rts)
%
kig(find(rts < eps*10^2)) = []; % Extract only positive values of interest
rts(find(rts < eps*10^2)) = []; % by nullifying negative & zero elements.
% plot(kig,rts)
%
% Extract first six mode values and compute cyclic frequency:
%
ubk = zeros(6,1);% Initialize the uniform beam k-vector.
%
for I = 1:6
 rts = sort(rts);
 ubk(I) = rts(I)/L;% Assign smallest remaining rts value to current mode.
 %
 % Nullify rts values that are close to or less than the current mode:
 %
 rts(find((rts-rts(I))<(eps*10^2))) = [];
end
%
wub = ubk.^2*sqrt(E*I/rho);% Note the L's already assumed unity.
fub = wub/(2*pi);
%
% © Using the eigenvalues, solve for the two remaining mode-shape
%   constants, and numerically evaluate the uniform beam mode-shapes:
%
ub_phi = zeros(length(xn),6); % Initialize uniform beam mode-shape matrix.
%
for I = 1:6
 k = ubk(I);
 %
 % Numerically evaluate boundary condition matrix using eigenvalue.
 %
 dmatn = [(-cos(k*L)+cosh(k*L)) (-sin(k*L)+sinh(k*L));...
        (sin(k*L)+sinh(k*L)),(-cos(k*L)+cosh(k*L))];
 %
 C3 = 1;                % Set value for C3 to unity.
 C1 = -(dmatn(2,2)*C3)/dmatn(2,1);% Solve for C1 using 2nd row of dmatn.
 %
 if abs(sum(dmatn*[C1;C3]))/max(abs([C1;C3])) > 10^(-4)
  warning = 'Uniform beam boundary condition solution may not be good.'
 end
 %
 % Evaluate and unit normalize mode-shape for solved boundary conditions:
 %
 msv = C1*(cos(k*xn)+cosh(k*xn))+C3*(sin(k*xn)+sinh(k*xn));
 ub_phi(:,I) = msv/max(abs(msv));
end
% ____________________________________________________________________
% Section (3)
% Create beam element mesh vector, generate finite element system
% matrices, and compute eigen system.
% DEFINE: snlv, fml, phi3, ncord3, phi20, ncord20, phi, ncord.
% USE: spatial, lden, lEI, lnbden.
% ..........................................................................
%
snlv = 1;% Due to the way <fem_mesh.m> was written, a value must be passed
%       for imposed node locations.  Setting it to one, imposes the end
%       of the beam, this triggers a warning, but <fem_mesh.m> then works.

nm = 20;% Highest number of finite elements to be employed.
fml = zeros(nm,6);% Initialize estimated frequency matrix.
%
for I = 1:nm
 nel = I;
 ncord = fem_mesh(spatial,lden,lEI,lnbden,snlv,nel);
 [Mfem,Kfem] = fem_form(spatial,lden,lEI,lnbden,ncord);
 [phi,fvn,rlab] = eigen_2o(Mfem,Kfem);% Compute, undamped eigen frequencies.
 %
 % Note: that with rigid body modes, warnings may be displayed regarding the
 % non-orthogonality of the mode-shapes, and a small imaginary content of the
 % zero frequencies is common.
 %
 % Extract first six bending modes.  (Note first two frequencies correspond
 % to zero-frequency rigid-body modes, so they are skipped.):
 %
 fbm = NaN*zeros(1,8);
 fbm(1:length(fvn)) = fvn'; % This leaves NaN's where no frequency is modeled.
 fml(i,1:6) = abs(fbm(3:8));% Estimated first six bending mode frequencies
 %                    in Hz. Some small imaginary content is common.
 %
 if nel == 3
  phi3 = phi;   % Save mode-shapes from 3 element approximation. This is
  ncord3 = ncord;% the first model to include the first 6 bending modes.
 end
end
%
%
 figure(1)
 clg
 set(gcf,'PaperOrientation','portrait');  % This series of commands configures
 set(gcf,'PaperUnits','inches');          % the plot window to effectively
 set(gcf,'PaperPosition',[1 1 3.5 4]);  % be incorporated into a report.
 set(gcf,'Units','inches','position',get(gcf,'PaperPosition'));
 set(gcf,'DefaultAxesFontSize',10);
 %
 plot([1:nm],fml(:,1),'k',[1:nm],fml(:,2),'k',[1:nm],fml(:,3),'k',...
    [1:nm],fml(:,4),'k',[1:nm],fml(:,5),'k',[1:nm],fml(:,6),'k')
 hold on
 plot([1:nm],fml(:,1),'ko',[1:nm],fml(:,2),'ko',[1:nm],fml(:,3),'ko',...
    [1:nm],fml(:,4),'ko',[1:nm],fml(:,5),'ko',[1:nm],fml(:,6),'ko')
 plot([1 nm],[fub fub],'k:')
 hold off
 av = axis;
 av(2) = 20;
 axis(av)
 set(gca,'YTick',fub)
 ylab = ' ';
 for I = 1:6
  ylab = str2mat(ylab, freq2str(fub(I)));
 end
 ylab(1,:) = [];
 set(gca,'YTickLabels',ylab)
 title('Finite Element Convergence to Mode Frequencies          ')
 ylabel('First Six Analytic Bending Modes                    ')
 xlabel('Number of Finite Elements')
 pos = get(gca,'position'); % This is in normalized coordinates
 pos(1)=pos(1) + pos(3)*.13; % Shift the subplot to the right.
 pos(3)=pos(3)*.85;       % Narrow it.
 set(gca,'position',pos);
 %
 nel = []; Mfem = []; Kfem = []; rlab = []; ylab = [];
% ____________________________________________________________________
% Section (6)
% Normalize and truncate the second order eigen modes.
% DEFINE: phi, fv, fvn, rlab.
% USE: ncord, ncord3, phi, phi3.
% ..........................................................................
%
%
ml_phi = zeros(length(xn),6);
ml_phi3 = zeros(length(xn),6);
ml_phi20 = zeros(length(xn),6);
%
for I = 1:6
 ml_phi(:,I) = mode_shape(ncord,spatial,phi(:,(I+2)));
 ml_phi(:,I) = ml_phi(:,I)/max(abs(ml_phi(:,I)));% Unit normalize.
 ml_phi3(:,I) = mode_shape(ncord3,spatial,phi3(:,(I+2)));
 ml_phi3(:,I) = ml_phi3(:,I)/max(abs(ml_phi3(:,I)));% Unit normalize.
end
%
%
%
if plot_on == 1
 figure(2)
 clg
 set(gcf,'PaperOrientation','portrait');  % This series of commands configures
 set(gcf,'PaperUnits','inches');          % the plot window to effectively
 set(gcf,'PaperPosition',[1 1 7 5]);  % be incorporated into a report.
 set(gcf,'Units','inches','position',get(gcf,'PaperPosition'));
 set(gcf,'DefaultAxesFontSize',10);
 n = 0;
 for j = 1:3
  for k = 0:3:3
   l = j+k;
```

```
272     n=n+1;
273     subplot(3,2,n)
274     plot(xn,ml_phi3(:,I),'k--',xn,ml_phi20(:,I),'k-.',...
275         xn,ml_phi(:,I),'k:',xn,ub_phi(:,I),'k');
276     title(['Bending Mode ' int2str(I) ', Analytic ' freq2str(fub(I))])
277     axis([-0.05 1.05 -1.05 1.05])
278     if n > 4
279       xlabel('Normalized Axial Position')
280     end
281     pos = get(gca,'position'); % This is in normalized coordinates
282     pos([4])=pos([4])*.9;        % Shrink the height by a factor of .9
283     pos([2])=pos([2]) + pos([4])*.1;% Raise the subplot by the saved height.
284     pos([3])=pos([3])*.95;       % Shrink the width by a factor of .9
285     if floor(n/2) == ceil(n/2)
286       pos([1])=pos([1]) + pos([3])*.05/2;% Shift the subplot to the right.
287     else
288       pos([1])=pos([1]) - pos([3])*.05/2;% Shift the subplot to the left.
289     end
290     set(gca,'position',pos);
291     if I == 3
292       legend('3 Element','20 Element',...
293            [int2str(nm) ' Element'],'Analytic');
294     end
295   end
296 end
297 subtitle('Uniform Beam Eigenvectors and Frequencies')
298 end
299 %
300 if print_on == 1
301   print -deps fig22.ps; % Print the file as an encapsulated Post-Script file.
302 end
303 %
```

7 BIBLIOGRAPHY

1. Dholiwar, D. K., "Development of a Hybrid Distributed-Lumped Parameter Openloop Model of Elevation Axis for a Gun System," Proceedings of the Seventh U.S. Army Symposium on Gun Dynamics, ARCCB-SP-93034, Benét Laboratories, Watervliet, NY, 11-13 May 1993, pp. 368-385.

2. Mattice, M. S., "State Space Model of the XM291 Tank Main Armament System," *Draft* Technical Report, U.S. Army ARDEC, AMSTA-AR-FSF-R, Picatinny Arsenal, NJ, 04 November 1994.

3. Gast, R., "Modal Analysis of the Dynamic Flexure in Tank Weapons," PhD Thesis, Rensselaer Polytechnic Institute, Troy, NY, May 1988.

4. Simulation of Barrel Dynamics: Users Manual, Danby Engineering, Cirencester, United Kingdom, 1992.

5. De Marchi, J. A., Ma, J., and Craig, K. C., "Experimental Degradation of Flexible Beam Control in the Presence of Drive-Train Non-Linearities," ASME Paper No. WAM-95-17, 1995.

6. Meirovitch, L., Elements of Vibration Analysis, McGraw-Hill, New York, NY, 1986.

7. Junkins, J. L., and Kim, Y., Introduction to Dynamics and Control of Flexible Structures, American Institute of Aeronautics and Astronautics, Inc., Washington, DC, 1993.

8. Leung, A. Y. T., Dynamic Stiffness and Substructures, Springer-Verlag, New York, NY, 1993.

9. Gast, R. G., "Normal Modes Analysis of Gun Vibrations by the Uniform Segment Method," Proceedings of the Fifth U.S. Army Symposium on Gun Dynamics, ARCCB-SP-87023, Benét Laboratories, Watervliet, NY, 23-25 September 1987, pp. 175-201.

10. Sennett, R. E., Matrix Analysis of Structures, Prentice-Hall, Englewood Cliffs, NJ, 1994.

11. Buchanan, G. R., Theory and Problems of Finite Element Analysis, Schaum's Outline Series, McGraw-Hill, New York, NY, 1995.

12. MATLAB® Reference Guide and MATLAB® User's Guide, The MathWorks, Inc., Natick, MA, July 1993.

13. Symbolic Math TOOLBOX User's Guide, The MathWorks, Inc., Natick, MA, August 1993.

14. Thompson, W. T., Theory of Vibration with Applications, Prentice Hall, Englewood Cliffs, NJ, 1993.

15. Dimarogonas, A. D., and Haddad, S., Vibration for Engineers, Prentice Hall, Englewood Cliffs, NJ, 1992.

16. Leipholz, H. H. E., and Abdel-Rohman, M., Control of Structures, Martinus Nijhoff Publishers, Boston, MA, 1986.

17. Shames, I. H., and Dyn, C. L., Energy and Finite Element Methods in Structural Mechanics, Hemisphere Publishing Corporation, New York, NY, 1985.

18. Kanchi, Madhu B., Matrix Methods of Structural Analysis, John Wiley & Sons, New York, NY, 1993.

19. Przemienieki, Theory of Matrix Structural Analysis, McGraw-Hill, New York, NY, 1968.

20. Meirovitch, L., Dynamics and Control of Structures, John Wiley & Sons, New York, NY, 1990.

21. Beyer, W. H. (Ed.), CRC Standard Mathematical Tables, CRC Press, Inc., Boca Raton, FL, 1984.

22. Brogan, W. L., Modern Control Theory, Prentice-Hall, Englewood Cliffs, NJ, 1985.

23. Scarton, H. A., Unpublished Course Notes, "Advanced Vibrations," Rensselaer Polytechnic Institute, Troy, NY, Fall 1993.

24. Friedland, B., Control System Design: An Introduction to State-Space Methods, McGraw-Hill, New York, NY, 1986.

25. Control System TOOLBOX User's Guide, The MathWorks, Inc., Natick, MA, June 1994.

26. Ogata, Katsuhiko, Modern Control Engineering, Prentice-Hall, Englewood Cliffs, NJ, 1970.

27. Frederick, D. K., and Chow, J. H., Feedback Control Problems: Using MATLAB® and The Control System Toolbox, PWS Publishing Company, Boston, MA, 1995.

28. Clancy, Tom, Armored Cav: A Guided Tour of an Armored Cavalry Regiment, Berkley Books, NY, 1994.

29. Ewins, D. J., Modal Testing: Theory and Practice, John Wiley & Sons, New York, NY, 1984.

30. Bracewell, R. N., The Fourier Transform and Its Applications, McGraw-Hill, New York, NY, 1986.

31. Zabar, Z., Levi, E., Birenbaum, L.,Vottis, P., Cipollo, M., and Kathe, E., "Pulsed Power to the Aid of Chemical Guns," Tenth IEEE International Pulsed Power Conference, Paper 5-3, Albuquerque, NM, 10-13 July 1995.

32. Kathe, E., Gast, R. G., Vottis, P. M., and Cipollo, M., "Analysis of Launch Induced Motion of a Hybrid Electromagnetic/Gas Gun," Eighth Electromagnetic Launch Symposium, Paper 109 B, Baltimore, MD, 21-24 April 1996.

33. Gast, R. G., "Curvature-Induced Motions of 60-mm Guns, Phase II: Modeling," ARDEC Technical Report ARCCB-TR-94002, Benét Laboratories, Watervliet, NY, January 1994.

34. Timoshenko, S., and Young, D. H., Vibration Problems in Engineering, D. Van Nostrand Company, New York, NY, January 1955.

TECHNICAL REPORT INTERNAL DISTRIBUTION LIST

	NO. OF COPIES
CHIEF, DEVELOPMENT ENGINEERING DIVISION	
ATTN: AMSTA-AR-CCB-DA	1
-DB	1
-DC	1
-DD	1
-DE	1
CHIEF, ENGINEERING DIVISION	
ATTN: AMSTA-AR-CCB-E	1
-EA	1
-EB	1
-EC	1
CHIEF, TECHNOLOGY DIVISION	
ATTN: AMSTA-AR-CCB-T	2
-TA	1
-TB	1
-TC	1
TECHNICAL LIBRARY	
ATTN: AMSTA-AR-CCB-O	5
TECHNICAL PUBLICATIONS & EDITING SECTION	
ATTN: AMSTA-AR-CCB-O	3
OPERATIONS DIRECTORATE	
ATTN: SIOWV-ODP-P	1
DIRECTOR, PROCUREMENT & CONTRACTING DIRECTORATE	
ATTN: SIOWV-PP	1
DIRECTOR, PRODUCT ASSURANCE & TEST DIRECTORATE	
ATTN: SIOWV-QA	1

NOTE: PLEASE NOTIFY DIRECTOR, BENÉT LABORATORIES, ATTN: AMSTA-AR-CCB-O OF ADDRESS CHANGES.

TECHNICAL REPORT EXTERNAL DISTRIBUTION LIST

	NO. OF COPIES
ASST SEC OF THE ARMY RESEARCH AND DEVELOPMENT ATTN: DEPT FOR SCI AND TECH THE PENTAGON WASHINGTON, D.C. 20310-0103	1
DEFENSE TECHNICAL INFO CENTER ATTN: DTIC-OCP (ACQUISITIONS) 8725 JOHN J. KINGMAN ROAD STE 0944 FT. BELVOIR, VA 22060-6218	2
COMMANDER U.S. ARMY ARDEC ATTN: AMSTA-AR-AEE, BLDG. 3022	1
AMSTA-AR-AES, BLDG. 321	1
AMSTA-AR-AET-O, BLDG. 183	1
AMSTA-AR-FSA, BLDG. 354	1
AMSTA-AR-FSM-E	1
AMSTA-AR-FSS-D, BLDG. 94	1
AMSTA-AR-IMC, BLDG. 59 PICATINNY ARSENAL, NJ 07806-5000	2
DIRECTOR U.S. ARMY RESEARCH LABORATORY ATTN: AMSRL-DD-T, BLDG. 305 ABERDEEN PROVING GROUND, MD 21005-5066	1
DIRECTOR U.S. ARMY RESEARCH LABORATORY ATTN: AMSRL-WT-PD (DR. B. BURNS) ABERDEEN PROVING GROUND, MD 21005-5066	1
DIRECTOR U.S. MATERIEL SYSTEMS ANALYSIS ACTV ATTN: AMXSY-MP ABERDEEN PROVING GROUND, MD 21005-5071	1
COMMANDER ROCK ISLAND ARSENAL ATTN: SMCRI-SEM ROCK ISLAND, IL 61299-5001	1
MIAC/CINDAS PURDUE UNIVERSITY 2595 YEAGER ROAD WEST LAFAYETTE, IN 47906-1398	1
COMMANDER U.S. ARMY TANK-AUTMV R&D COMMAND ATTN: AMSTA-DDL (TECH LIBRARY) WARREN, MI 48397-5000	1
COMMANDER U.S. MILITARY ACADEMY ATTN: DEPARTMENT OF MECHANICS WEST POINT, NY 10966-1792	1
U.S. ARMY MISSILE COMMAND REDSTONE SCIENTIFIC INFO CENTER ATTN: AMSMI-RD-CS-R/DOCUMENTS BLDG. 4484 REDSTONE ARSENAL, AL 35898-5241	2
COMMANDER U.S. ARMY FOREIGN SCI & TECH CENTER ATTN: DRXST-SD 220 7TH STREET, N.E. CHARLOTTESVILLE, VA 22901	1
COMMANDER U.S. ARMY LABCOM, ISA ATTN: SLCIS-IM-TL 2800 POWER MILL ROAD ADELPHI, MD 20783-1145	1

NOTE: PLEASE NOTIFY COMMANDER, ARMAMENT RESEARCH, DEVELOPMENT, AND ENGINEERING CENTER, BENÉT LABORATORIES, CCAC, U.S. ARMY TANK-AUTOMOTIVE AND ARMAMENTS COMMAND, AMSTA-AR-CCB-O, WATERVLIET, NY 12189-4050 OF ADDRESS CHANGES.

TECHNICAL REPORT EXTERNAL DISTRIBUTION LIST (CONT'D)

	NO. OF COPIES
COMMANDER U.S. ARMY RESEARCH OFFICE ATTN: CHIEF, IPO P.O. BOX 12211 RESEARCH TRIANGLE PARK, NC 27709-2211	1
DIRECTOR U.S. NAVAL RESEARCH LABORATORY ATTN: MATERIALS SCI & TECH DIV WASHINGTON, D.C. 20375	1
WRIGHT LABORATORY ARMAMENT DIRECTORATE ATTN: WL/MNM EGLIN AFB, FL 32542-6810	1
WRIGHT LABORATORY ARMAMENT DIRECTORATE ATTN: WL/MNMF EGLIN AFB, FL 32542-6810	1

NOTE: PLEASE NOTIFY COMMANDER, ARMAMENT RESEARCH, DEVELOPMENT, AND ENGINEERING CENTER, BENÉT LABORATORIES, CCAC, U.S. ARMY TANK-AUTOMOTIVE AND ARMAMENTS COMMAND, AMSTA-AR-CCB-O, WATERVLIET, NY 12189-4050 OF ADDRESS CHANGES.